SAND97-2902
Unlimited Release
Printed December 1997

Distribution
Category UC-705

Final Report for the Robustness-Agile Asynchronous Transfer Mode (ATM) Encryption Laboratory Directed Research and Development Project

Thomas D. Tarman, Robert L. Hutchinson, Peter E. Sholander,
and Richard J. Granfield
Network Systems Surety Department

Lyndon G. Pierson
Advanced Networking Integration Department

Perry J. Robertson
Advanced Devices and Applications Department

Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-0449

Edward L. Witzke
RE/SPEC Inc.
Albuquerque, NM 87110

Abstract

A Robustness-Agile Encryptor (RAE) provides confidentiality services for each Asynchronous Transfer Mode (ATM) virtual circuit at an arbitrary strength (or cryptographic robustness). This enhances ATM's flexibility, particularly in cases where political or policy constraints limit the choices for encryption algorithms. It may also reduce operational costs, since several users can share one RAE. However, this flexibility raises the concern that unauthorized personnel may use inappropriate algorithms (particularly the "stronger" algorithms). Hence, RAE requires strong use-control techniques (such as cryptographic authentication) in order to enforce each site's security policy. This report describes Sandia's implementation of a prototype RAE. It covers issues such as use-control, cryptographic synchronization, high-speed crypto-module design and RAE's effect on ATM Quality of Service (QoS).

Acknowledgments

The authors wish to express their gratitude to the following individuals for their help in this project: Craig Wilcox for his work on the DES encryption logic and PLD11, Larry Pucket and Terry Hardin for their work on the design and layout of the PLD11, Jeff Ingle and Mark Bean of NSA for their collaboration, and Mike Sjulin and Mike Vahle for their support of this work.

Contents

Figures

1. Introduction

Asynchronous Transfer Mode (ATM) is a data communications technology which is becoming increasingly important for both local and wide area networks. One reason for this emergence is that ATM is very flexible in terms of architecture and application support. Architectural flexibility is achieved through ATM's suitability for both local area networks and wide area networks. This architectural flexibility allows ATM to support application data "end-to-end", without expensive protocol conversion or protocol inefficiencies (in contrast with the Transmission Control Protocol, which is inefficient in high bandwidth-delay environments). Flexible application support is achieved by ATM because it uses small, fixed-length "cells" as its basic unit of information, which allows for fine-grained multiplexing of data, thereby supporting many different applications simultaneously (e.g., email, file transfers, telephony, and video). In addition, this fine-grained multiplexing also provides statistical multiplexing gains, which increases link utilization. This is particularly important when using expensive links, such as long-haul or satellite data communications links.

As ATM matures, users now want confidentiality services for their ATM applications. Those services must provide strong confidentiality, yet still maintain ATM's flexibility. In response to user needs, standards organizations (most notably, the ATM Forum) are indeed developing flexible ATM security specifications. However, actual products which exercise this flexibility do not currently exist. As such, this project developed one such flexible encryption device – namely a Robustness Agile Encryptor (RAE).

This report describes the results of a Sandia-funded Laboratory-Directed Research and Development (LDRD) project entitled "Robustness-Agile ATM Encryption and Associated Use Control". A *Robustness-Agile Encryptor* (RAE) allows ATM confidentiality services to be provided for each virtual circuit at an arbitrary level of strength (or *cryptographic robustness*). This helps to maintain ATM's flexibility, particularly in cases where political or policy constraints limit the choices for encryption algorithms. However, along with this flexibility comes the concern that certain algorithms (particularly the "stronger" algorithms) may be used by unauthorized personnel. Hence, strong *use-control* techniques (such as cryptographic authentication) are required to ensure that encryption algorithms are only used by those who are authorized by the site's security policy.

This project had two purposes. The first was to research several issues associated with robustness-agile ATM encryption. The second was the development of prototype robustness-agile ATM encryptors, and their associated use-control mechanisms. The issues explored by this project (and documented in this report) include algorithm switching and other architectural aspects, ATM QoS effects, authentication and key exchange protocols, and encryptor use-control issues. This project did not finish building a complete prototype encryptor. However, it did successfully develop, or model, many sub-components. In particular, it showed algorithm agility, strong access control to the algorithms, and the effects of algorithm agility on ATM QoS. The completed sub-components include ATM signaling-based *security agents*, a smart card-based use-control mechanism, cryptomodules that use a programmable logic implementation of the Data Encryption Standard (DES), and hardware for switching ATM cells to appropriate cryptomodules.

This report describes in detail the issues in robustness-agile ATM encryption, and the design of a prototype robustness-agile ATM encryption device. Section 2 motivates robustness-agile encryption technology by describing how it can be used to provide security for ATM networks in a flexible manner. Section 3 describes in detail some of the issues associated with the implementation of robustness-agile encryptors, including ATM QoS effects, the need for authentication and use-control, throughput, and synchronization. A more detailed analysis of the ATM QoS effects is provided in Section 4, along with a relevant example. The design and development of the prototype encryptor developed in this project is described in Section 5. Finally, conclusions are presented in Section 6.

2. Motivation for Robustness-Agile ATM Encryption

As stated in [8], different applications often have different security requirements. These security requirements may be dictated by the expected lifetime of the application data, the real-time nature of the data (which may preclude slow cryptographic operations such as file encryption), and site security policy. In order to apply the appropriate level of confidentiality protection to an application data stream, the encryption service must be *robustness agile*.

The terms *key-agile*, *algorithm-agile*, and *robustness-agile* (defined in [8]), refer to the ability of an encryption device to apply different *encryption contexts* (i.e., keys, algorithms, and algorithm/key lengths, respectively) to each application data circuit. In the case of ATM, an application data circuit is identified by the Virtual Path Identifier (VPI) and the Virtual Channel Identifier (VCI) for a given link. Therefore, robustness agile encryption requires that the encryptor be capable of applying an algorithm and a key of appropriate length to cells on a specified VPI and VCI.

With ATM, the Virtual Channel Connections (VCCs) which carry application data can originate from an end system, or they can be aggregated at a private switch port which is either connected to another private switch, or to a public carrier. By placing a robustness agile encryptor between the host and the first switch (on the User to Network Interface, or UNI), or alternatively, embedding the encryptor within the host's Network Interface Card (NIC), VCCs can be encrypted directly at the source. This approach is advantageous if one does not trust other users in the private network, or if one wishes to form Virtual Private Networks (VPNs) of subsets of hosts which attach to a common physical network. These alternatives, embedded NIC encryption and UNI encryption, are shown in Figure 1 as the "Embedded RAE" and the "UNI RAE", respectively.

Alternatively, the RAE may be placed at the Private Network to Network Interface (PNNI), and/or at the Public UNI. By placing the RAE at the PNNI, the RAE provides encryption services for VCCs which terminate at hosts on the plaintext side of the encryptor. This approach offers significant cost savings because one encryptor is shared among many users. However, this approach is only useful if the common ATM network and its users are trusted not to reveal the plaintext data. This is generally true if the network and hosts are physically protected, and the users are screened and trained according to appropriate site security policies. This concept of *workgroup encryption* can be taken further by placing the RAE at the Public UNI, which affords encryption services for an entire site. Placement of the RAE at the PNNI and Public UNI is also shown in the reference model in Figure 1.

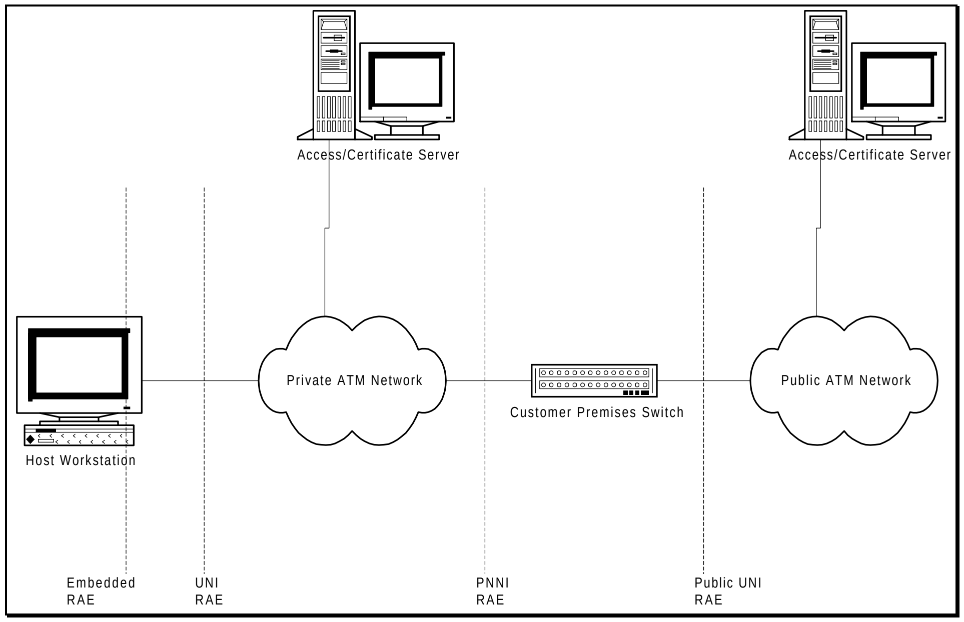


Figure 1: RAE Reference Model

By placing the encryptor at the only point of entry into the network (either the site network, or a private subnetwork), the opportunity exists to control remote access into the protected network. In order to perform this access control, one or more *Access Control Servers* and/or *Certificate Servers* are required. When considering the role of these two servers, one must consider the authentication/authorization function. In order to grant permissions to a requesting entity (such as a remote node which wishes to connect to a protected host), the authorizing device must identify the requesting entity (authentication), and determine that entity's permissions according to an *Access Control List* (ACL). While a number of authentication techniques exist (e.g., passwords, PINs, etc.), cryptographically strong authentication requires the requesting user to participate in an authentication protocol, and requires the authorizing device to validate the requestor's identity using this protocol. The latter function requires that the authorizing device has access to the requester's public key.

The role of the Access/Certificate servers shown in Figure 1 is to provide the encryptor with the ACL and the requester's public key when the requester attempts to connect to a protected node. By providing this information from a centralized location, management of ACL and public key information is simplified, as multiple encryptors can simply query a "well known" server when they wish to perform authorization functions. Although these servers provide encryptors with the ability to properly authorize access to remote hosts, these servers can also be used to determine which internal users have access to which algorithms in the encryptor. This additional function is described further in Section 3.

It should be noted that this two-step authorization process (authentication and ACL lookup) can be condensed into a single step. New public key certificate formats (such as the Simple Public Key Infrastructure, or SPKI [6]) include the access control credentials in the certificate. When such certificates are examined by an authorizing device, it does not need to query an ACL to determine the entity's access privileges.

As stated earlier in this section, application traffic may have a variety of security protection requirements. These requirements are dictated by the nature of the traffic, and by the *security policy* of the site or entity which owns the data. In an open ATM network which allows hosts to establish VCCs to many other sites, the encryptor must be flexible in order to apply the appropriate protection to the application VCC, and to be able to interoperate with the encryptor at the remote host. This scenario is shown in Figure 2.

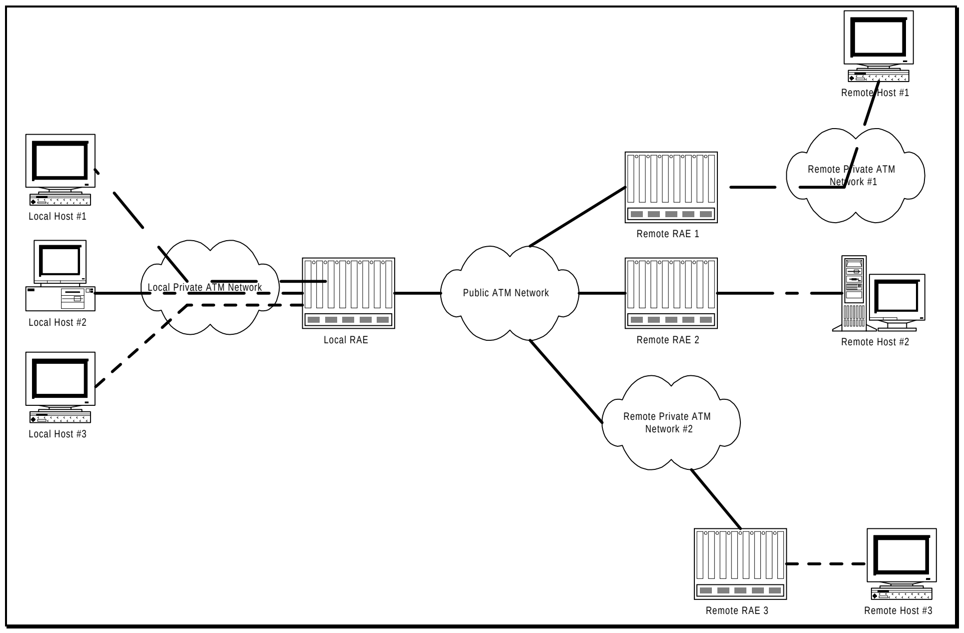


Figure 2: Example Network using RAEs

Differences in security policy at the various sites may exist for a number of reasons. First, each site may have their own algorithms which they prefer for technical reasons (such as the desire for low encryption latency or increased perceived strength). Second, sites may prefer certain algorithms for financial reasons (e.g., cost of royalties). Finally, sites may be constrained to only use certain algorithms due to political reasons (e.g., export and/or import laws). At any rate, different sites may in general have different security policies, which may lead to differences in algorithms, key lengths, or both. In order to establish ATM VCCs to a variety of sites, the ability to support a variety of algorithms is required.

The ability to support a variety of algorithms may be provided in two ways, either by using a “crypto pool” which contains a number of separate encryption devices, or by using a single robustness-agile encryptor. However, the first approach is infeasible because it requires VCCs to be routed according to the required security attributes for the VCC. This form of security policy-based routing does not yet exist in the ATM standards. Furthermore, the first approach is more costly in terms of system management because each device may have its own peculiarities, requiring additional training and staff in order to properly operate the “crypto pool”. However, by using a single RAE, both of these problems are eliminated.

Nevertheless, when several encryption algorithms (each of which having different strengths and key lengths) are provided in a single ATM encryptor, a number of implementation issues arise. These issues are described in the following section.

3. Implementation Issues in Robustness-Agile Encryption

As described in [8], the implementation of key-agile, algorithm-agile and robustness-agile encryptors (of which the RAE is the super-set) carries with it a number of issues. These issues occur because the RAE's functions, while similar in many respects to an ATM switch, are nevertheless rather unique to ATM networking.

For reasons stated below, the key-agile ATM encryption process is considered to resemble the ATM switching process. In particular, key-agile encryptors are similar to two-port ATM switches. For comparison, ATM switches modify cell headers and switch cells based on the "switching context" associated with each VPI/VCI. The initial association of switching information with a virtual circuit may be a manual operation for Permanent Virtual Circuits (PVCs). The initial association might also occur automatically at connection setup time for Switched Virtual Circuits (SVCs). Once this association is established, for each incoming cell, the ATM switch performs an associative lookup of switching information, based on the VPI/VCI found in each cell's header. This switching information maps the incoming VPI/VCI into the appropriate outgoing VPI/VCI. It also conditions the hardware to switch the cell out the proper port.

Key-agile ATM encryptors resemble ATM switches in the sense that they modify cells based on the "context" associated with each VPI/VCI. However, ATM encryptors modify the *payloads* of the cells, rather than the *headers*. As with switches, the initial association of the cryptographic variables and state with each virtual circuit may be a manual operation or be performed at SVC connection setup time (or later) via the methods invoked for key management [11]. Once a cryptographic context is established for a virtual circuit, for each incoming cell the encryptor performs an associative lookup of the cryptographic context based on the VPI/VCI found in each cell's header. The encryptor then uses that cryptographic context to transform the incoming cell payload (plaintext or ciphertext) into the appropriate outgoing payload (ciphertext or plaintext). Finally, the encryptor typically routes the cell out the opposite port of a two-port device (although single-port, or so-called *one-armed encryptors* are not precluded). Hence key-agile ATM encryptors resemble a two-port ATM-switch. Algorithm-agile and robustness-agile ATM encryptors also resemble a two-port ATM-switch; however, they have an additional complication since their internal paths and data structures may depend on the context information.

All three types of agile-encryption have some common issues. First, the number of potential contexts (VPI/VCIs) and the amount of information per context may both be large. In general, there may be either 2^{24} possible UNI VPI/VCI combinations or 2^{28} possible NNI VPI/VCI combinations. Hence, implementing the cryptographic-context lookup with "straight indexed" (flat) memory may be costly. Because the number of simultaneously active contexts is likely to be small, an efficient key-agile encryptor could use an associative memory lookup to determine the key and other cryptographic state information associated with each cell stream. Clearly, encryption algorithms that must associate larger keys and greater state information will be more cumbersome (expensive) to implement than algorithms that require a minimum of key and state information. In either case, large content addressable memories (or the even larger sequential memories required) with access times on the order of ATM cell header processing times are expensive and/or unavailable. Hence, until large, inexpensive and fast content addressable memories do become available, current designs compromise either the virtual circuit space over which circuits can be encrypted, or the cell processing latency, or both.

Another common issue is synchronization. When an encryptor and decryptor pair have lost synchronization, the decrypted data stream is scrambled, which leads to excessive data loss. While some cryptographic algorithms or modes of operation are "self synchronizing", others require both initial synchronization and resynchronization after each cell loss. Since each virtual circuit has independent synchronization, the synchronization state information adds to the amount of information that must be associatively maintained for each encrypted virtual circuit.

A third common issue is throughput. If the encryption or decryption process cannot keep up with the maximum possible cell arrival rate, then the cell traffic throughput on that virtual circuit must be throttled in some fashion to avoid cell loss. This can be done via *Call Admission Control* (CAC) at virtual circuit setup time (for Constant, Variable, and Unspecified Bit Rate traffic) or by participation in the flow control after virtual circuit setup (for Available Bit Rate traffic). In either case, the encryption/decryption devices must participate in the establishment and/or control of the VC, making these devices no longer "transparent" to the switching network.

Algorithm-agile and robustness-agile encryption adds additional complexity since the virtual circuits can use different encryption algorithms, modes of operation, and (in the case of robustness-agile encryption) key lengths. These choices add more data to be associatively maintained for each virtual circuit. In addition, the various algorithms may require varying amounts of key material and state information, and may also add additional hardware for optional processing of the cell payloads.

Algorithm-agile and robustness-agile encryptors can indirectly affect the ATM Quality-of-Service (QoS), since different algorithms may have different delay, throughput, error magnification, and/or sensitivity to synchronization upset. In that case, the ATM QoS negotiation which occurs at connection setup must incorporate knowledge of the delays of the encryption methods contained in the encryptor. One interesting effect is due to the different relative latencies of each algorithm. Algorithm-agile and robustness-agile encryption preserves cell order within an individual virtual circuit and among virtual circuits that use the same encryption algorithm. However, such encryption may cause re-ordering of cells that use different encryption algorithms. This reordering may introduce additional *Cell Transfer Delay* (CTD) and/or *Cell Delay Variation* (CDV) among cells of the same virtual circuit which use the same encryption algorithm. Until the encryptors participate in the QoS negotiation, algorithm-agile encryptors may need to append output buffering to low latency algorithms. This delay-equalization, with the higher latency algorithms, trades lowered CDV for increased CTD. If algorithm-agile encryptors can participate in QoS negotiations, then the set of available algorithms may be dynamically restricted to maintain a previously negotiated CDV bound. This topic is addressed in more detail in Section 4 and in [10].

4. Analysis of QoS Effects of Robustness-Agile Encryption

An interesting issue is the effect of algorithm-agile and robustness-agile encryption on ATM QoS, particularly on Cell Transfer Delay (CTD) and Cell Delay Variation (CDV). Algorithm-agile encryptors come in two basic varieties – namely a single shared-processor or multiple parallel-processors. Performance analysis for the single shared-processor case is a classic scheduling-problem. However, a single, shared-processor architecture may not scale to Gigabit per second ATM link-rates because of the context-switching overhead between different encryption algorithms. Hence, this project's QoS study (documented in detail in [10]) considered the multiple parallel-processor architecture shown in Figure 3. In this architecture, the *cell sorter* forwards cells to the appropriate encryption/decryption pipeline. Each pipeline is then a key-agile encryptor that implements one encryption/decryption algorithm. Finally, the *output queue* re-combines cells after encryption/decryption.

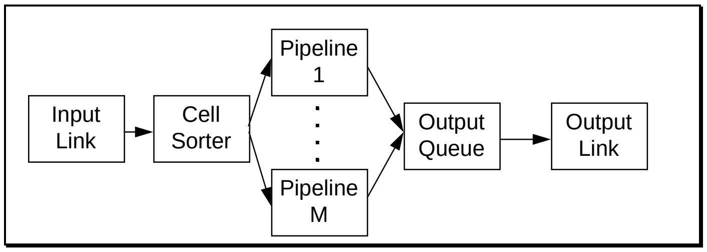


Figure 3: Parallel Pipeline Architecture for an Algorithm-Agile Encryptor

Three important ATM Quality of Service (QoS) parameters are CTD, CDV and Cell Loss Ratio (CLR). As defined in [2], CDV is the difference between the best-case and worst-case CTD, where the best-case is the fixed network-delay and the worst-case is the CTD that is exceeded with a user-defined probability α. "Excessive" CTD is undesirable for applications, such as voice, web-browsing and interactive game-playing, that have response-time requirements. "Excessive" CDV is also undesirable because it complicates timing recovery for Constant Bit-Rate (CBR) applications such as video, by requiring "smoothing" buffers that contribute to excessive cell transfer delay. CDV can also cause buffer overflows, and hence cell-loss. The ATM network and the ATM user negotiate mutually acceptable QoS values for a particular VC, during call setup.

In both the single processor and multiprocessor cases, the encryptor is a *single-input*, *single-output* (SISO) device that serves one input ATM-link and one output ATM-link. So, the encryptor can guarantee near-zero cell loss. However, ATM QoS is still an issue with algorithm-agile (and robustness-agile) encryption because each encryption algorithm, within an algorithm-agile encryptor, may have a different latency, or per-cell encryption time. This differing latency may re-order cells, between ATM *Virtual Circuits* (VCs), and hence cause CDV. Consistent with the ATM standards, algorithm-agile encryption does preserve cell-order within an individual VC. Furthermore, algorithm-agile encryptors also maintain cell ordering within all VCs that use the same encryption algorithm. However, such encryptors may re-order cells that use different encryption algorithms. For example, let the cells from VCs 1 and 2 use encryption algorithms 1 and 2, respectively. Let algorithm 1 have a longer encryption time, per cell, than algorithm 2. Then, for example, an input cell-sequence of 1,1,2,2,... might produce the output cell-sequence 2,2,1,1,... . This introduces CDV into the output cell-stream. Both a single, shared-processor architecture and the multiple, parallel-processor architecture can generate cell reordering, and hence CDV. The interaction between this encryption-induced CDV and the ATM traffic-management policies (such as Usage Parameter Control (UPC), or policing) negotiated for the affected VC can cause added cell-loss. Hence, an algorithm-agile encryptor should participate in the QoS negotiations during ATM call-setup. That participation requires models for the encryption-induced CDV.

This project's QoS study [10] derived the following results when the queueing discipline at the output queue is First-In, First-Out (FIFO):

> Assuming that all pipelines can process (i.e., encrypt or decrypt) cells at the offered rate, let T_i be the cell processing time for encryption pipeline i, divided by the cell's link transmission time (therefore, T_i is measured in cells). Also, let T_1 and T_m be the cell processing times for the longest and shortest encryption pipelines, respectively. In that case, the CTD is bounded by the sum of the longest pipeline's execution time, the output queuing time and the output transmission time.
>
> $$\text{CTD} \leq \lceil T_1 \rceil + 1 \text{ cells} \qquad (1)$$
>
> Furthermore, there is no CDV generation if:
>
> $$| T_i - T_j | < 1, \forall\ i \neq j \qquad (2)$$
>
> If Equation 2 holds, then the differential delay through the encryptor's different pipelines does not produce cell re-ordering. If Equation 2 does not hold, then cell re-ordering can occur. In that case:
>
> $$\text{CDV} \leq \lfloor T_1 - T_m \rfloor \text{ cells} \qquad (3)$$
>
> In practice, an encryptor often has a bypass path, with T_m nearly equal to zero. So, a simpler CDV bound omits T_m.

As an example, assume an algorithm-agile encryptor in which pipeline 1 implements the DES algorithm, and pipeline 2 implements Sandia's scalable ATM encryption algorithm [9]. Also assume that there is no bypass path. Finally, assume that pipeline 1 contains 16 stages, with 64 bits per stage, and that the cell link transmission time $t_c \approx 2.8\ \mu S$/cell. (That transmission time corresponds to SONET OC-3.) In that case, the cell storage in pipeline 1 is:

$$T_1 = \frac{(16 / 6)t_c}{t_c} = 2.667 \text{ cells}$$

where the numerator denotes the pipeline delay (16 stages divided by 6 stages per (384-bit) cell, multiplied by the time per cell), and the denominator denotes the normalization factor.

Similarly, assuming that Sandia's scalable ATM encryption algorithm can encrypt a cell in 2.7 μS [9], then

$$T_2 = \frac{2.7 * 10^{-6}}{t_c} = 0.964 \text{ cells}$$

where the numerator again denotes the algorithm delay, and the denominator denotes the normalization factor.

Since condition (2) is not met, the CTD and CDV bounds are as follows:

$$\text{CTD} \leq 4 \text{ cells} = 11.2\ \mu S$$

and

$$\text{CDV} \leq 1 \text{ cell} = 2.8\ \mu S$$

These CTD and CDV values are similar to those for a typical ATM switch. Therefore, as with ATM switches, this algorithm-agile encryptor should participate in the QoS negotiations. Otherwise, these effects could cause traffic contract violations, and hence cell loss.

In addition to the FIFO case, the project also studied equalizing the pipeline delays (e.g., through "build-out"). As was expected, the CTD for that case was equal to the longest pipeline delay, while the CDV was zero.

These two cases (FIFO queueing versus delay equalization) illustrate an interesting tradeoff. In the FIFO case, the mean CTD is lower than that for the "build-out" case because some of the ATM cells traverse the lower-delay paths. However, in the FIFO case, the CDV is greater than the "build-out" case due to these differential delays through the various encryption pipelines. So, if the encryption pipeline delays (i.e., before "build-out") are relatively similar, then one might use delay equalization so as to minimize CDV. Alternatively, if the relative difference between the algorithm delays is great, then one might choose the FIFO approach in order to minimize the mean CTD.

5. Design and Development of a Prototype ATM Robustness Agile Encryptor

5.1 Development RAE System Architecture

Early in this project, a “development RAE system architecture” was designed. This network architecture was developed for two reasons: 1) to provide an environment for development of software and hardware modules, and 2) to provide an environment to support functional and performance tests on the various modules once they were completed. This development architecture is shown in the following figure:

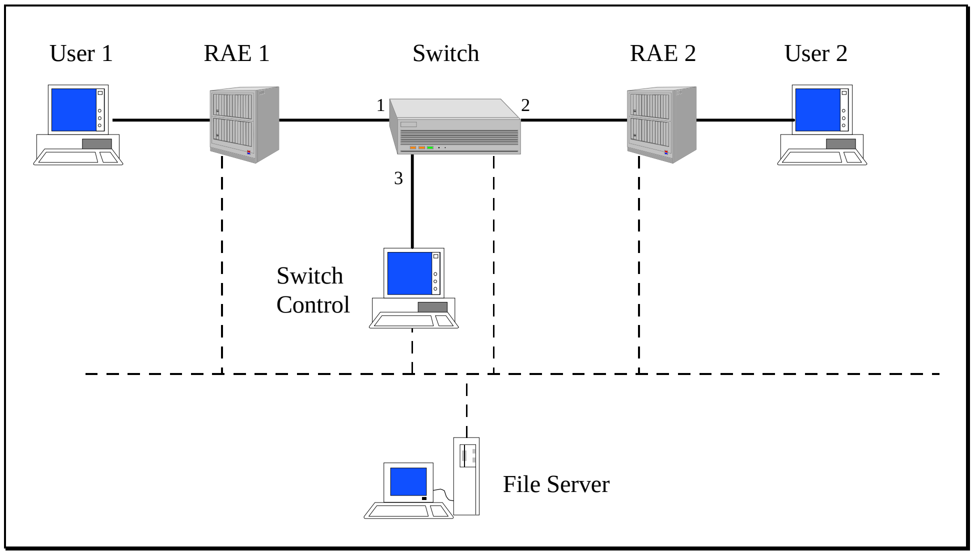


Figure 4: Development RAE Architecture

In this architecture, the two robustness-agile encryptors are located at the UNI, and therefore encrypt ATM cells for individual hosts rather than for groups of hosts. These encryptors are configured with two algorithms – DES (with 56 bit effective key), and Sandia’s fast, scalable ATM encryption algorithm developed under a previous project [9]. This configuration was selected because it paired a very popular, but slow algorithm (DES) with an extremely fast algorithm to purposefully amplify the RAE’s effects on QoS.

A VMEbus SPARC processor was selected to control the encryptor hardware, and to perform signaling-based authentication, key exchange, and access control. Since the processor boards were configured as diskless workstations, they were provided with Ethernet connections to allow them to mount a file server which contains the development software. Although “production” RAE devices may not be designed this way, this architecture was ideal for developing and testing new software modules.

The purpose of the switch is to simulate an ATM “cloud” between the two RAEs. Although the switch does not implement a security agent, modifications to the signaling protocol software were performed in order to support the transport of Security Services Information Elements during connection setup. Finally, the switch was assigned a separate switch control processor which implements the IISP signaling protocol on all ports.

All devices, except for the switch, implement security agents in this system. These security agents participate in the three security associations shown in Figure 5: one association between Host 1 and RAE 1, another association between Host 2 and RAE 2, and a third association between RAE 1 and RAE 2. The security associations between the users and their respective RAEs allows for mutual authentication to occur, and hence, strong access control to the algorithms in the encryptor. The security association between the two RAEs allows for mutual authentication, and the exchange of ATM cell encryption keys.

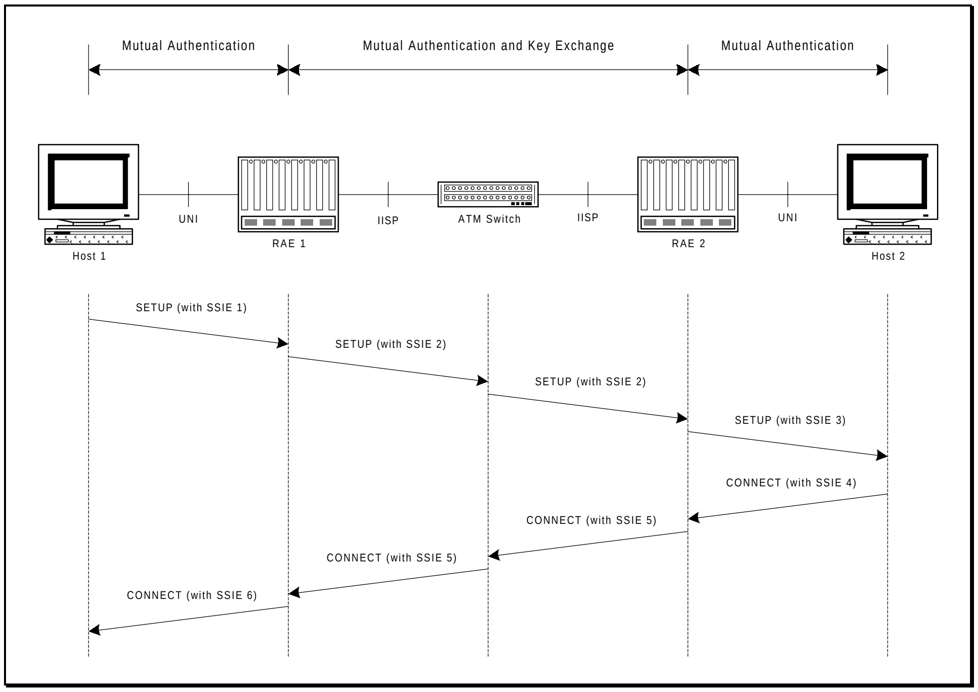


Figure 5: Security Message Exchange

Although the ATM Forum’s Security Specification [1] describes a number of options for security message exchange, the security exchange protocol selected for this implementation was the two-way, signaling-based security message exchange. This protocol was selected because it could be developed separately from the hardware, unlike the in-band protocol option, which required hardware to forward cells on the virtual circuit to the security agent. In addition, because this protocol option is a two-way message protocol, it fit gracefully into signaling’s setup/connect SVC establishment protocol.

5.2 RAE Prototype Architecture

The RAE system architecture is shown in Figure 6. The RAE has the following major components: the *shell*, the *cryptomodules*, the *control processor* (which implements signaling, access control, and hardware control), and the *key management module*.

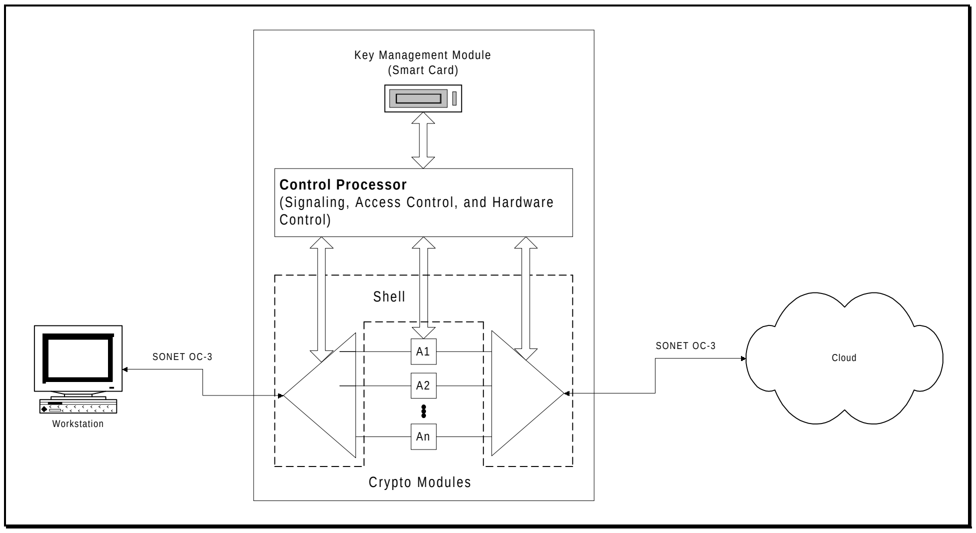


Figure 6: RAE Prototype Architecture

The shell is the RAE logic which implements the algorithm-neutral functions. As cells enter the plaintext or the ciphertext ports, the shell determines which algorithm shall process each cell, according to the contents of the shell context memory which contain (VPI, VCI, algorithm) tuples for each active virtual circuit. Once this determination is made, the shell routes the ATM cell and its cryptographic index to the appropriate cryptomodule for encryption or decryption. Finally, as the cell leaves the cryptomodule, it is combined with the output cell streams from the other cryptomodules, and is queued for transmission on the output SONET link. Depending on the configuration of the RAE, the cell can either be queued for First Come First Served (FCFS) queueing, or it can be buffered so that the delays across all algorithm pipelines are matched (i.e., *build-out*). The shell design and functions are described further in Section 5.2.1.1.

The algorithm and mode-specific processing functions (i.e., encryption and decryption) are performed in the cryptomodule. When a cell arrives at the input of the cryptomodule, the cryptomodule retrieves the encryption or decryption key from its context memory, using the cryptographic index provided by the shell. Depending on the mode of operation that the cryptomodule implements, it may also need to look up additional encryption context. For example, if the cryptomodule implements DES in Cipher Block Chaining mode, then it will need to look up the previous ciphertext block in order to "chain" it into this block of encryption. Once the key and context values are determined for this ATM cell, it is processed using the designated algorithm and mode. When the encryption or decryption is completed, the cryptomodule stores the current encryption/decryption context so that it will be available when the next cell arrives on the same virtual circuit, and it sends the cell to the shell to be queued for transmission. The design of the cryptomodule is described in Section 5.2.1.2.

In order to achieve key-length agility, our design supports the use of variable-width Linear Feedback Shift Registers (LFSRs) as key generators. This approach allows for a fixed key-length algorithm such as Triple-DES to be "dialed-down" to a desired level of robustness.

As stated earlier in this report, the RAE must control user access to a given encryption algorithm at a given level of robustness (or strength). Furthermore, the RAE must support the negotiation of algorithms and parameters with another RAE. These functions are performed by the signaling software, which runs on the control processor. This software implements an ATM Forum "Security Agent" for node authentication, and uses the ITU's standard Q.2931 signaling protocol for security message transport. When a connection is requested by an end system, the security agent performs the necessary authentication steps, consults an access control table which maps user identities to cryptographic robustness levels, and determines whether the connection is authorized. In addition, if the RAE interacts with another RAE, then the security agents must exchange data encryption keys for the new connection. More details on the Q.2931 signaling software and security agent can be found in Sections 5.2.2.1 and 5.2.2.2, respectively.

Once the secure connection negotiation is successful, the hardware control software on the control processor configures the shell and crypto module hardware according to the security association. The control processor selects the correct crypto module, loads keys, and enables the connection. In setting up the connection, the control processor loads the cryptographic key into the crypto module and a key index into the shell. Each RAE has one shell, which acts as a limited ATM switch. The shell switches connection setup (ATM *Control Plane*) cells to the control processor, and switches cells on an established connection (ATM *User Plane*) to the correct crypto module. The shell also performs OAM and specified VPI/VCI diagnostic loop back. The control processor can configure the shell to discard cells on a specified VPI/VCI. The control processor can also configure the shell to discard all cells on unknown VPI/VCI pairs. Each RAE has one or more crypto modules, each of which is capable of at least one encryption/decryption algorithm. In addition, the crypto modules are key agile. The shell switches a cell to the crypto module and indexes the correct cryptographic key. Once switched to the crypto module, the cell is either encrypted or decrypted, then delivered back to the shell and finally scheduled for transmission. More details on the hardware control functions can be found in Section 5.2.2.3

The key management module allows for the secure storage of authentication keys and the exchange of data encryption keys. In addition, the cryptographic algorithms which are used for these services are also implemented in this module. In this implementation, a smart card and card reader are used to perform these functions. The smart card-based implementation of the key management module is described further in Section 5.2.2.4.

5.2.1 RAE Hardware

5.2.1.1 Shell

The RAE shell acts like a limited ATM switch. Figure 7 shows the allowed switching paths. Received cells can be switched to the control processor, any crypto module, or the loop back path. Note that the shell does not have a bypass path. If bypass is required, cells must be switched to a crypto module which implements a bypass mode. The shell does not have the ability to pass plain text to the cipher text network without going through a crypto module. Cells processed by the crypto modules are combined with loopback cells and cells from the control processor. These cells are scheduled for transmission by a configurable cell scheduler. The shell can be configured to discard cells with a specific VC or unknown VC. This allows flexibility in security policies.

The shell can be logically divided into three main sections: control, transmit engine, and receive engine. First, a brief description of the process that creates a secure connection is provided. Next, the details of the three main shell sections is discussed. Finally, a description of the shell as a system of these three sections is provided.

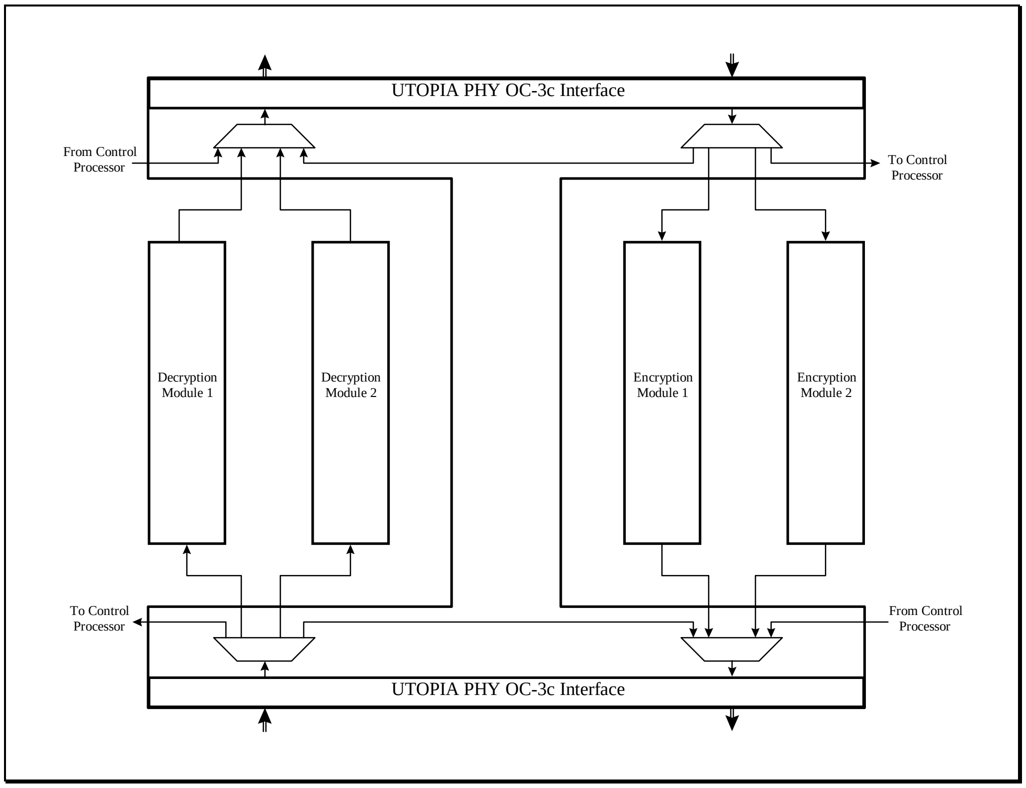


Figure 7: RAE Switching Paths

5.2.1.1.1 Secure connection establishment

Once the connection establishment is complete and the security association is formed, the control processor must configure the shell and crypto module to enable the connection. The control processor selects a crypto module that supports the necessary algorithm. It creates an index and stores a cryptographic key in a location on the crypto module pointed to by the index. Next, it creates an entry in the shell's switching table. The entry contains VPI/VCI information for the incoming cell along with a crypto module designation to which the cell will be switched. The entry also contains the index that points to the key in the crypto module. At this point, the secure connection is enabled. The shell uses the VPI/VCI of incoming cells to determine if a secure connection exists. If the VPI/VCI of a received cell match an entry in the switching table, the cell is switched to the crypto module specified in the table. The shell also sends the index that points to the correct key. The crypto module receives the cell and uses the index to retrieve the correct key. It then encrypts or decrypts the cell payload. Next, the processed cell is delivered back to the shell. Finally, the shell schedules the cell for transmission and sends it when the channel capacity and that VC's priority allow. A secure connection is torn down by removing the entry in the shell's switching table.

5.2.1.1.2 Control

The control processor communicates with the shell and crypto modules over a VMEbus. The shell and crypto modules are slave devices. The shell uses a Cypress slave VME interface chip set (SVIC). At power up, the SVIC is programmed using a serial PROM. The serial PROM data specifies A32/D32 and A32/D16 transfer modes. Interrupts are available but unused. The control processor uses the VMEbus to insert and extract connection setup messages, configure the shell, and configure the crypto modules.

The shell has a 16 bit control register and a 16 bit status register. The control register responds to two different addresses. Writing a ‘1’ to the first address will set the associated control bit. Writing a ‘0’ to the second address will clear the associated control bit. Either address may be read without clearing any control bits. The status register is read only It also responds to two different addresses. Reading address one will not clear any status bits that are set. Address two is clear on read. Figure 8 shows the control register definition. Figure 9 shows the status register definition. Note that bits which are not explicitly defined are reserved.

Control Register Defintion

bit	function	default
0	Enable PT Receive Engine	1
1	Enable PT Transmit Engine	1
2	Enable CT Receive Engine	1
3	Enable CT Transmit Engine	1
4	Reset IgT 2 (RXPT/TXCT)	0
5	Reset IgT 1 (RXCT/TXPT)	0
6	Discard PT cells on unknown VC	1
7	Discard CT cells on unknown VC	1
8	PT Loop Back (all cells)	0
9	CT loop back (all cells)	0
10	Master Reset	0

All control bits use positive logic, 1 = true

Figure 8: Control Register Definition

Status Register Defintion

bit	indication
0	PT Cell Available
1	RXPT FIFO overrun
2	RXPT LOS
3	RXPT OOF
4	CT Cell Available
5	RXCT FIFO overrun
6	RXCT LOS
7	RXCT OOF
8	
9	

All Status bits use positive logic, 1 = true

Figure 9: Status Register Definition

When a connection setup cell (VPI/VCI = 0/5) is received by the shell on either the plain text or cipher text network, one of two status register bits is set by the shell, and the associated counter is also incremented. The status register bit indicates that at least one connection setup cell is available. The counter indicates the number of available connection setup cells. The control processor must poll the status register and extract connection setup cells as they become available. The shell does not perform any ATM processing and therefore does not delineate messages; this is the responsibility of the control processor.

By setting or clearing bits in the control register, the control processor can enable/disable the plain text and cipher text transmit and receive engines. It can also define the security policy for cells on unknown virtual circuits. A bit in the control register determines whether cells on an unknown virtual circuit are discarded or switched to the default crypto module. The control register also has self clearing bits that perform a master reset of the board or individually reset the UTOPIA (Universal Test and Operations Physical Interface for ATM) interfaces. Lastly, the control register has two bits that control plain and cipher text diagnostic loop back.

The status register contains SONET (Synchronous Optical NETwork) indications such as loss of signal. The status register also has bits to indicate the availability of connection setup cells. If the cell available bit is set, a clear-on-read counter indicates the number of available cells. The shell is designed to keep up with a full SONET OC-3c data rate. If an error occurs and any cell is lost, an error indication bit in the status register is set.

5.2.1.1.3 Cell Switching Table

The format of the switching table is shown in Figure 10. An inbound cell's VPI/VCI is decoded into an index into the switching table. The switching table contains a three bit field (board/algorithm) which specifies the destination port for this cell. Bit 15 specifies whether the table entry is valid. If bit 15 is clear, the control register bit "discard unknown VCs" determines whether this cell is passed to the default crypto module or discarded. On reset of the RAE, the non-real-time controller will clear bit 15 in all switching table entries. Setting bit 14 causes this cell to be discarded. All other bits are ignored if bit 14 is set. Setting bit 13 causes this cell to be looped back. Bits 12:10 specify the intended output port (crypto module). Bits 9:7 are reserved for future use. Bits 6:0 contain the index that point to the crypto module's key.

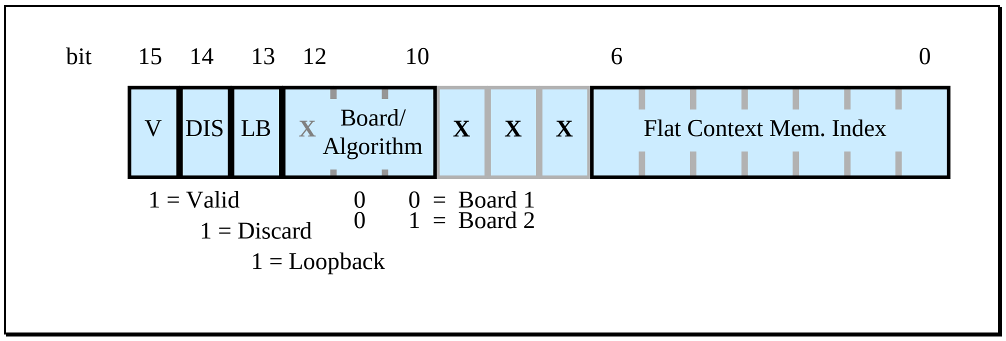


Figure 10: Switching Table Entry

The current shell implementation does not fully decode the VPI/VCI. The least significant nine bits of VCI and the least significant four bits of the VPI are combined to create a 13 bit index. Our initial requirement was to provide a full VPI/VCI decode to create a non-aliasing index into the switching table. We identified two alternatives to achieve this: use enough switching memory to fully cover the entire 24 bit space, and use content addressable memory (CAM). As shown in Figure 10, each switching table entry requires two bytes of data. To cover the entire 24 bit space, we would need 32 MB (24 bits = 16MB x 2B per entry) of memory. Because of the high cost and low density of static random access memory (SRAM), board space and cost considerations forced us to consider dynamic random access memory (DRAM). DRAM is roughly a factor of ten times slower than SRAM and requires a refresh cycle which further slows access by limiting availability. For these reasons, a DRAM switching table does not scale well. We also considered content addressable memory (CAM). This type of memory can be SRAM-based. All 24 bits of a VPI/VCI would be stored in an available CAM location whose address is A_i. The switching table entry (format shown in Figure 5) would be stored in standard SRAM at location A_i. When a new cell arrives, its VPI/VCI would be used to "address" the CAM. That is, the arrived VPI/VCI would be compared against all entries in the CAM. If a match is found, the CAM will return A_i. Then A_i would be used to address the switching table entry in standard SRAM. The CAM would be sized to support the maximum number of simultaneous connections. This solution is scaleable and can be cost effective. We chose to implement a simple dual port SRAM memory configuration due to time constraints. Note that the rest of the shell design does not assume any specific memory format, therefore the shell's switching memory can be upgraded with little impact on the rest of the design.

5.2.1.1.4 Shell Memory Map

The non-real-time processor uses a VMEbus to communicate with the shell and crypto modules. Figure 11, Figure 12, and Figure 13 specify the addressing of the shell. The most significant 12 bits of the address are used to select the different boards. Different boards include the shell and crypto modules. The next four bits are used to select the region on the board. Figure 12 shows the different regions for the Shell and crypto modules (PLD-11). The least significant 16 bits are used for addressing within each region. Figure 13 shows the address space in each of the Shell board's regions.

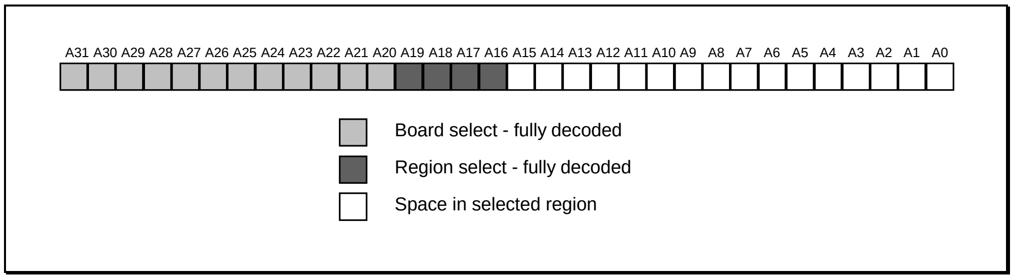
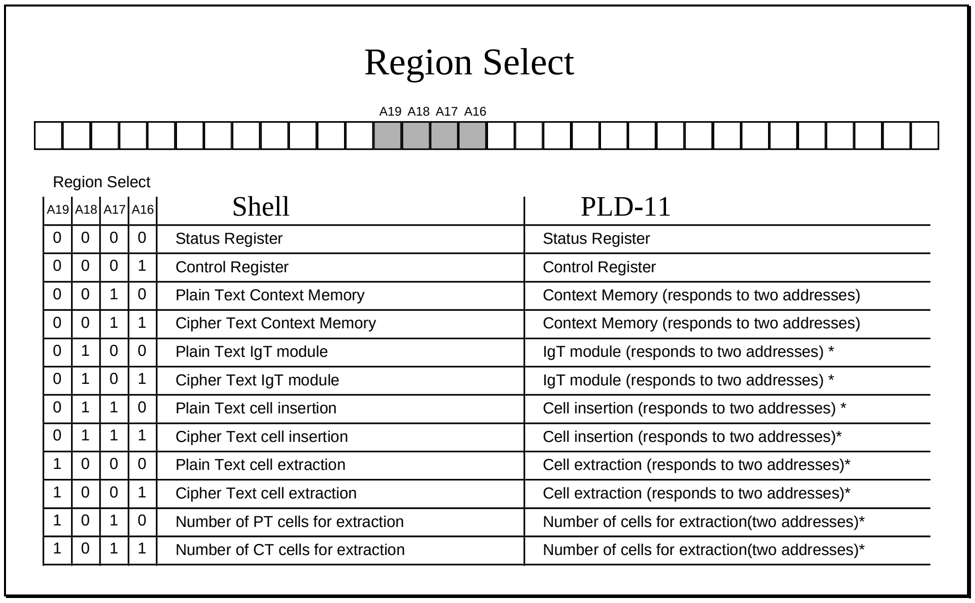

A31 A30 A29 A28 A27 A26 A25 A24 A23 A22 A21 A20 A19 A18 A17 A16 A15 A14 A13 A12 A11 A10 A9 A8 A7 A6 A5 A4 A3 A2 A1 A0
Board select - fully decoded
Region select - fully decoded
Space in selected region

Figure 11: Shell Addressing

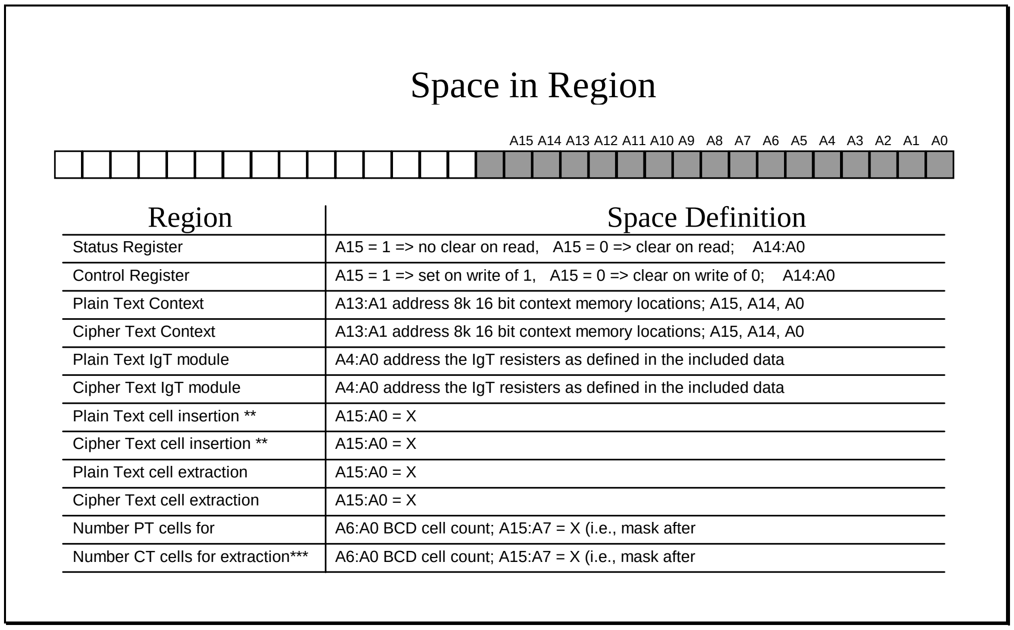

A19 A18 A17 A16

Region Select

Region Select

A19	A18	A17	A16	Shell	PLD-11
0	0	0	0	Status Register	Status Register
0	0	0	1	Control Register	Control Register
0	0	1	0	Plain Text Context Memory	Context Memory (responds to two addresses)
0	0	1	1	Cipher Text Context Memory	Context Memory (responds to two addresses)
0	1	0	0	Plain Text IgT module	IgT module (responds to two addresses) *
0	1	0	1	Cipher Text IgT module	IgT module (responds to two addresses) *
0	1	1	0	Plain Text cell insertion	Cell insertion (responds to two addresses) *
0	1	1	1	Cipher Text cell insertion	Cell insertion (responds to two addresses)*
1	0	0	0	Plain Text cell extraction	Cell extraction (responds to two addresses)*
1	0	0	1	Cipher Text cell extraction	Cell extraction (responds to two addresses)*
1	0	1	0	Number of PT cells for extraction	Number of cells for extraction(two addresses)*
1	0	1	1	Number of CT cells for extraction	Number of cells for extraction(two addresses)*

Figure 12: Region Definitions


A15 A14 A13 A12 A11 A10 A9 A8 A7 A6 A5 A4 A3 A2 A1 A0

Space in Region

Region	Space Definition
Status Register	A15 = 1 => no clear on read, A15 = 0 => clear on read; A14:A0
Control Register	A15 = 1 => set on write of 1, A15 = 0 => clear on write of 0; A14:A0
Plain Text Context	A13:A1 address 8k 16 bit context memory locations; A15, A14, A0
Cipher Text Context	A13:A1 address 8k 16 bit context memory locations; A15, A14, A0
Plain Text IgT module	A4:A0 address the IgT resisters as defined in the included data
Cipher Text IgT module	A4:A0 address the IgT resisters as defined in the included data
Plain Text cell insertion **	A15:A0 = X
Cipher Text cell insertion **	A15:A0 = X
Plain Text cell extraction	A15:A0 = X
Cipher Text cell extraction	A15:A0 = X
Number PT cells for	A6:A0 BCD cell count; A15:A7 = X (i.e., mask after
Number CT cells for extraction***	A6:A0 BCD cell count; A15:A7 = X (i.e., mask after

Figure 13: Space Definitions

5.2.1.1.5 Transmit Engine

The RAE has two transmit paths: plaintext transmit and ciphertext transmit. Each transmit path accepts cells from three sources: cells processed by a crypto module, loop back cells, and cells injected by the non-real-time-processor. The job of the transmit engine is to queue these cells and schedule them for transmission.

The interface between the transmit engine and the crypto modules is 16 bits wide. This allows a slower clock to be used on transfers between the shell and crypto modules. Cells are transferred from the crypto module using the format defined in UTOPIA II [5]. The third 16 bit transfer contains user-defined data and is discarded (see [5]). The transmit engine performs a width adaptation to convert the 16 bit UTOPIA II format to an 8 bit UTOPIA I format. The resultant cell is queued in a FIFO dedicated to its source and scheduled for transmission.

The transmit engine accept cells from each cell source and buffers them in individual first-in-first-out (FIFO) memory. A flexible cell scheduler monitors each of the cell sources and determines when a cell is available for transmission. If more than one cell is available for transmission, the scheduler decides which cell should be transmitted next according to a programmable scheduling policy. A serial programmable read only memory must be updated to support new scheduling policies. The existing policy gives priority to cells received from the crypto modules and schedules those cells in the order received. Cells from the loop back path are given second priority and cells from the non-real time processor are given third priority. The scheduler assigns a designator to each of the cell sources. A cell is scheduled for transmission by storing the designator in a scheduling FIFO memory. The transmit engine determines if the scheduling FIFO contains a designator. If it does, the transmit engine reads the designator and determines which source is represented. The transmit engine then transfers the designated cell into the UTOPIA interface where the cell is framed and transmitted.

Bits 1 and 3 in the control register enable the plain text and cipher text transmit engines respectively. If these bits are clear, (that is, ‘0’), cells from all sources are discarded.

5.2.1.1.6 Receive Engine

The shell has two receive paths: plaintext and ciphertext. Cells received from the UTOPIA interface can be switched to one of three main paths: crypto module path, loop back path, and the non-real-time path. In addition, these cells can be discarded. The receive engine switches cells by examining the VPI and VCI in the cell header. Bits 0 and 2 in the control register enable the plaintext and ciphertext receive engines when set. Both receive engines are enabled when the board is reset.

Cells from the UTOPIA interface are buffered long enough to extract the VPI and VCI from the header and make a switching decision. The least significant four bits of the VPI are concatenated with the least significant nine bits of the VCI to form an address into the switching table. Since the entire VPI/VCI space is not decoded, aliasing is possible. That is, more than one VPI/VCI pair can generate the same address into the switching table. Therefore, this design relies on sequential generation of VPI/VCI pairs by the ATM switch. If more than one connection generates the same switching table address, cells from these connections will be mixed together. Therefore, the control processor must ensure that when a new virtual circuit is enabled, that the switching table address of that virtual circuit does not conflict with a previous one.

Once the switching table address is formed, it is used to read the contents of the switching table. The receive engine uses fields in the switching table to switch the cell. The first field is used to indicate the validity of the entry. If the entry is invalid, the receive engine tests the state of a bit in the control register to determine if the cell should be discarded or switched to the default crypto module or a bypass path. The second field is used to instruct the receive engine to discard cells on this VPI/VCI. The third field contains the port number to which the cell must be switched. The fourth field contains an index which will be passed to the crypto module to address the correct cryptographic key.

If a cell is to be switched to a crypto module, the cell must be converted from UTOPIA I format to UTOPIA II format. The receive engine performs this width and format conversion.

5.2.1.1.7 Logic Development

The shell board's logic was developed on an Altera workstation. The design is hierarchical with graphical description files at the top levels and text description files at the lower levels. The design is divided into the three major modules previously defined: VME control, transmit engine, and receive engine. Timing and functional simulations on the design were performed to determine design correctness.

5.2.1.2 Cryptomodule

The cryptographic module is implemented on a Sandia-developed Flexible Programmable Logic Board (PLD11). This circuit board was designed to be applied to the prototyping of several other high speed communication functions as well as to the prototype cryptographic module in this research effort.

First, the PLD11 circuit board design will be described in general. The sections following will then describe the logic compiled into the programmable logic devices to accomplish the cryptographic module function.

5.2.1.2.1 PLD11

The PLD11 board provides a method of programming over a million logic gates to process a wide (over 384 bits) data path for high speed throughput. The programming can be accomplished in VHDL (Very High Speed Integrated Circuit (VHSIC) Hardware Description Language) and compiled for downloading into the board via a serial interface. The board is flexible in that it can be reprogrammed quickly and easily to accomplish any given task.

The PLD11 board is a 9U VME board that contains 11 Altera 10K100 Programmable Logic Devices (PLDs) and associated support electronics. The board contains devices to connect the board to a standard VME interface. It also contains clock distribution, Universal Asynchronous Receiver/Transmitter (UART) and serial programmable support. The board was designed to provide a very wide interface (384 inputs, and 384 outputs) for high throughput applications. The components on the board are capable of operation as high as 100 MHz resulting in a throughput of 38.4 Gbps. The Altera 10K100 devices have approximately 100,000 programmable gates. The entire board then acts as a single, large PLD having 768 I/O and 1.1 million programmable logic gates. In addition, the board is uniquely designed to accommodate a patchwork of boards, each interconnected to provide expandability either in serial fashion for greater number of gates, or in parallel for greater throughput. The middle three PLDs are suited for performing additional logic or switching 96 bit wide data buses between alternate destinations.

The 14.44” X 15.75” PLD board contains 11 Altera 10K100 devices. The board also contains interface circuitry to a VME bus, 50 MHz crystal oscillator, clock distribution, serial interface and 155 Mbps UTOPIA I ATM interface. The board was autorouted using Veribest software. The digital lines on the board are designed to be about 75 ohm, 5 mil traces. This impedance was chosen to reduce reflections between PLDs.

A logical flow of the board is shown in Figure 14.

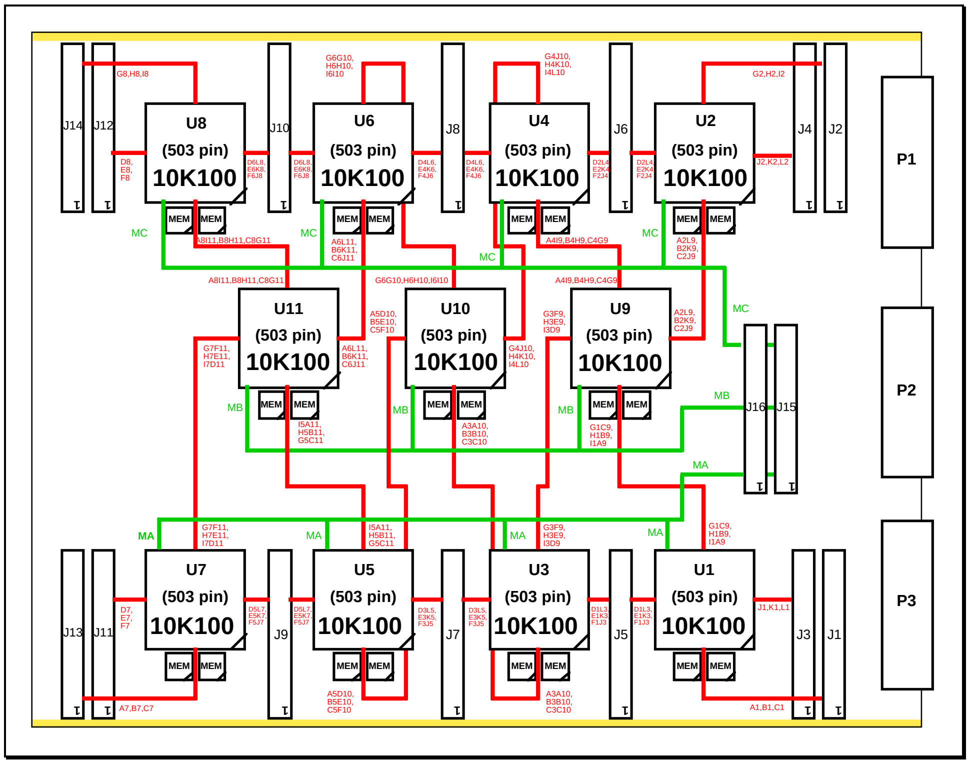


Figure 14: PLD11 Logical Signal Flows

The layout of the board is shown in Figure 15.

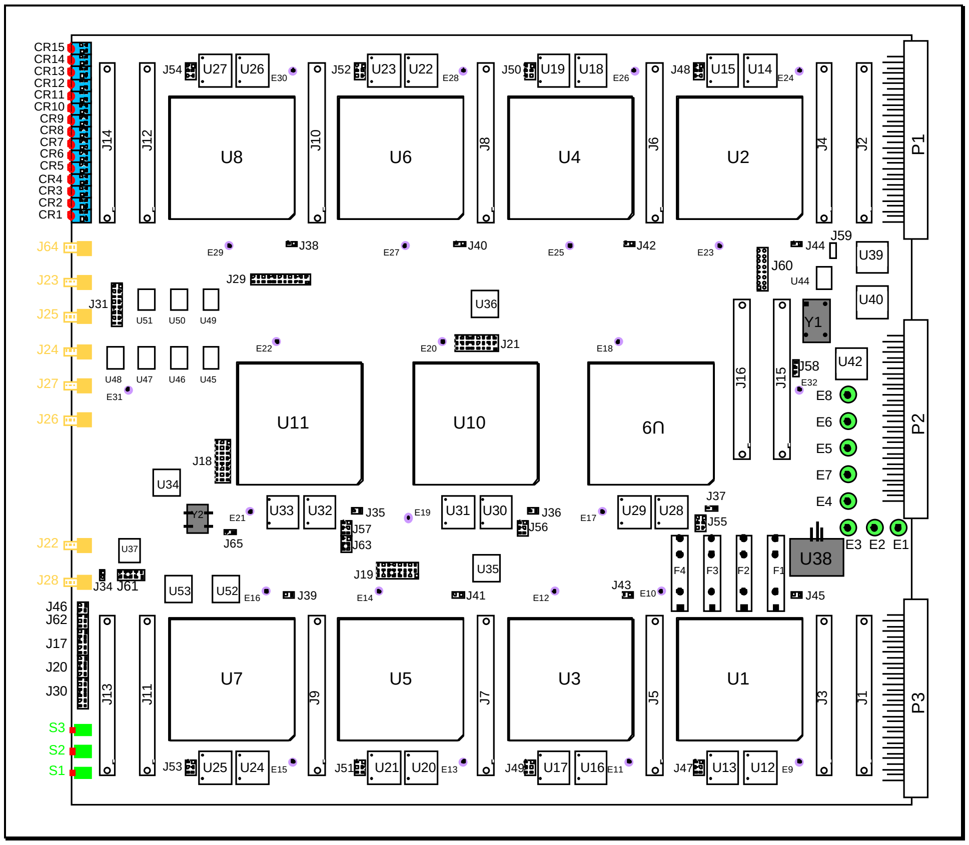


Figure 15: PLD11 Board Layout

The PLD11 board contains a clock distribution circuit that is capable of delivering the reference clock, in phase to all Altera 10k100 devices on the board with very little clock skew. In addition, it has been designed to allow for the adjusting of the skew as well as the phase and frequency of the clock. The clock distribution is shown Figure 16.

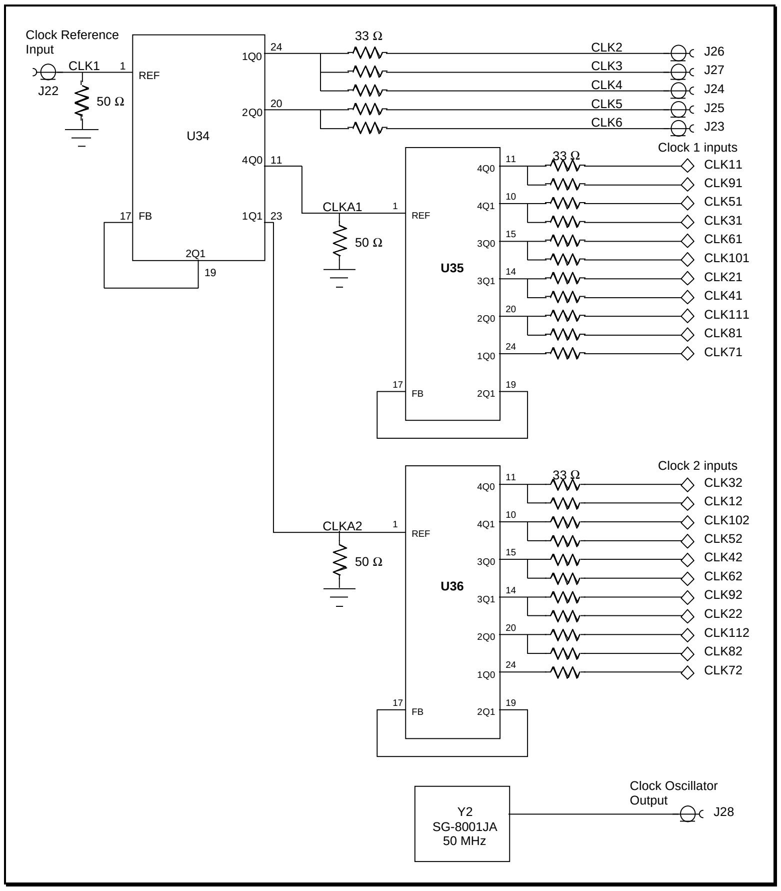


Figure 16: PLD11 Clock Distribution Network

Commercially available PLD devices do not have either the wide data path or the large number of programmable gates needed for many aggressive applications. This board provides both. In addition, the board has data paths which can be rerouted using the middle three PLD devices for even greater flexibility. This rerouting can switch two 96 bit wide buses between alternate destinations. This function enables the board to perform context switching for ATM applications. This board is capable of supporting 4 independent 96 bit data streams continuously through the board at the clock rate.

This board is an improvement over existing technology in that existing PLD devices do not have the large number of I/O pins or the large gate count required to accomplish large tasks. The board’s architecture allows for the use of multiple boards, in either a serial or parallel fashion, to increase the number of gates or increase the throughput without limit to meet application requirements.

The board has applications to a number of projects within Sandia and numerous other outside applications. For example, the board will be used as a testbed for several applications related to high speed modem development for the Air Force Phillips Lab. These applications include the study of communications protocols, digital demodulator architecture studies and the study of military applications of commercial low earth orbit satellite systems. Within Sandia, this board is used by this project, and by the Scaleable ATM Encryption (SAE) project to develop demonstration hardware. The board also can be used for development of Public Key Cryptography, Bit Error Rate test equipment, image filtering acceleration, SAR Radar image processing, and real time CMOS chip emulation.

This technology development can be applied to a number of government and industry applications. It is a low cost alternative to the development of specialized hardware for any application requiring both a large number (approx. 1 million) of programmable gates and high speed operation (greater than 50 MHz). Commercial applications might include CMOS chip emulation. Military applications include image processing and radar image analysis. Also, the board can be used in commercial ATM encryption equipment.

5.2.1.2.2 Cryptomodule Design Issues

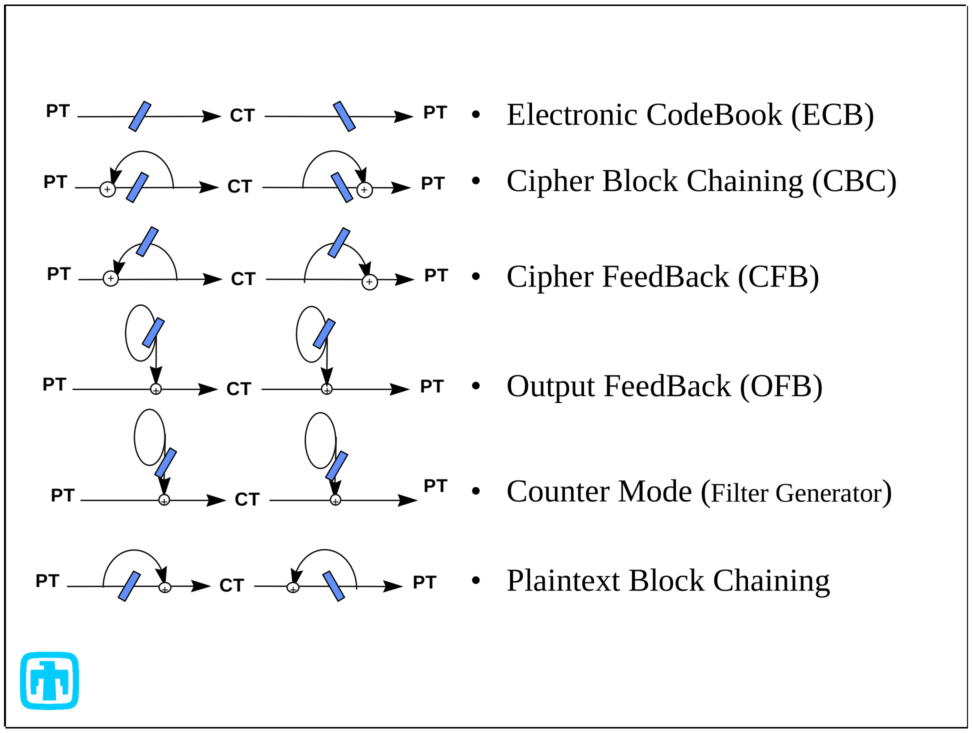
Networks of massively parallel processors and visualization workstations will require ATM encryption, SONET encryption, and “non-SONET” encryption in the range of 1 to 100 Gb/s.

SONET encryption will be required at OC-48c (2.5 Gb/s) and OC-192c (10 Gb/s) in order to protect against traffic analysis in some systems. Even though ATM encryption does not by itself protect against traffic analysis, an even greater need is anticipated for ATM encryption at OC-48c and OC-192c in order to reduce the number and cost of encryption units required for protection of communications.

The scaling of variable bit rate ATM encryption appears to be harder than the scaling of SONET encryption. The fastest encryptors built today operate in the range of 0.05 to 0.622 Gb/s. Current ATM encryption prototypes and products implement ATM encryption at OC-12c (0.622 Gb/s) or lower rates. There exists a need to continue active R&D to scale “raw encryption” speeds far past those required for OC-12 in order to achieve the 1000x speedup required for the efficient remote utilization of MPP computing resources..

This research project has focused on Algorithm & Robustness Agility, Modes of Operation, and Synchronization with lesser emphasis on speed and implementation cost.

Some applications require not only agility of the algorithm (the nonlinear block cipher used), but also agility of the “feedback mode of operation”. Several “feedback modes of operation” are depicted in Figure 17. The simplest mode is the Electronic Code Book (ECB) mode, but this mode allows repeating patterns in the plaintext to be also discerned in the ciphertext, enabling some statistical cryptanalysis. The use of other “feedback modes” have evolved to present certain cryptographic advantages. Figure 18 tabulates some of the advantages and disadvantages of the more common modes of operation. Of special interest is the “counter mode” since it can be scaled by parallel implementation to operate at very high data rates, and yet can interoperate with less expensive lower speed encryptors or decryptors implemented with a lesser scale factor . Counter mode involves a “cryptographic state” which prevents inference of patterns in the plaintext from patterns in the ciphertext. This provides protection against “dictionary lookup” and “playback attacks” which is not provided by ECB mode. One disadvantage of counter mode is that it is not “self-synchronizing”. That is, a method of synchronizing the decryption process with the encryption process must be designed into a counter mode cryptosystem. Even though Cipher Block Chaining (CBC) mode cannot be scaled to high speeds and yet interoperate with dissimilarly scaled implementations, it has an advantage for low cost implementation at low speeds because of its automatic self-synchronizing property, while still protecting against “dictionary lookup” and “playback” attacks.



- Electronic CodeBook (ECB)
- Cipher Block Chaining (CBC)
- Cipher FeedBack (CFB)
- Output FeedBack (OFB)
- Counter Mode (Filter Generator)
- Plaintext Block Chaining

Figure 17: Summary of Feedback Modes of Operation

Mode	Security	Implementation	Fault Tolerance	Crypto Sync
ECB	- plaintext patterns are not concealed	+ no feedback + no IV storage + encryption and decryption are parallelizable	+ cell loss has no additional negative effects - ciphertext error magnification	+ self synchronizing
CBC	+ plaintext patterns are concealed	- feedback from encryption output - IV storage - encryption is not parallelizable + decryption is parallelizable	+ cell loss causes 1 additional block of plaintext to be corrupted - ciphertext error magnification	+ self synchronizing
CFB	+ plaintext patterns are concealed	- feedback from encryption output - IV storage - encryption is not parallelizable + decryption is parallelizable	+ cell loss causes 1 additional block of plaintext to be corrupted - ciphertext error magnification	+ self synchronizing
OFB	+ plaintext patterns are concealed	- feedback from encryption output - IV storage - encryption and decryption are not parallelizable	- cell loss causes loss of crypto synchronization + no ciphertext error magnification	- requires periodic resynch
Counter	+ plaintext patterns are	- feedback from encryption input - IV storage + encryption and	- cell loss causes loss of crypto synchronization + no ciphertext error magnification	- requires periodic resynch

Figure 18: Characteristics of Common Modes of Operation

After examining the properties of the many “feedback modes of operation” that can be used with any nonlinear block encryption algorithm, this research suggests that a single mode of operation does not span the entire “cost vs. performance” space. At least two modes of operation which protect against dictionary lookup and playback will ultimately prevail. These are 1) the counter mode for its ability to scale and interoperate, despite its synchronization requirement, and 2) CBC mode for its ability to be synchronized at lower speeds at less cost, despite its inability to scale and interoperate. The “cost of synchronization” vs. “scalability” of these two modes are contrasted in Figure 19.

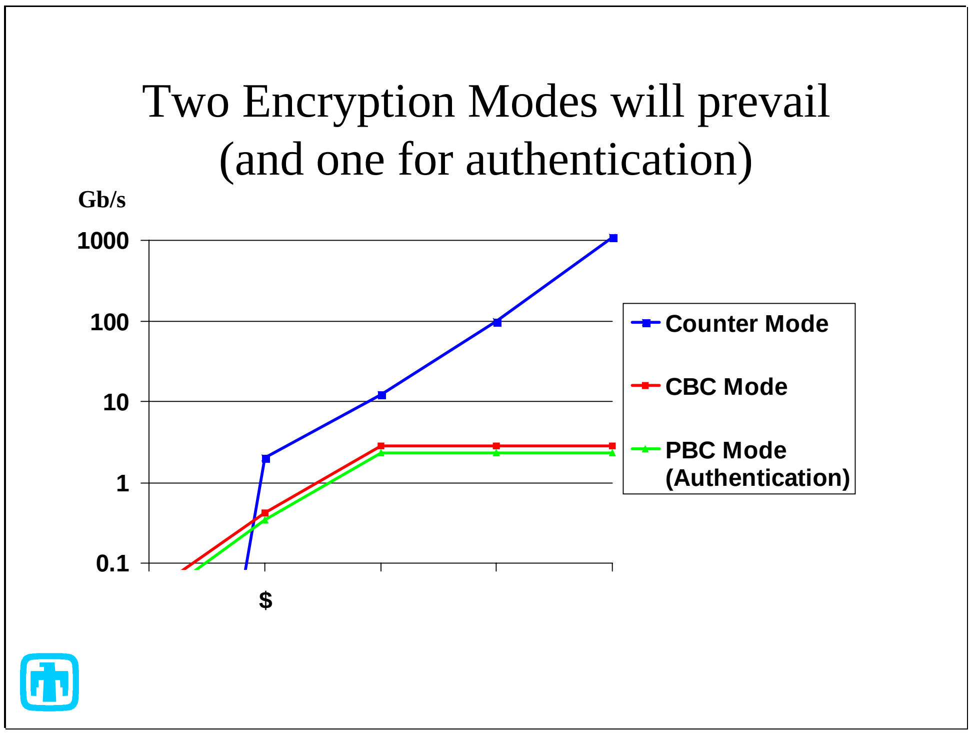


Figure 19: "Cost of Synchronization" vs. "Scalability" for Counter Mode and CBC

An encrypted link is usually not as reliable as an unencrypted link because 1) it involves extra equipment in the path 2) the equipment is mysterious and easy to blame, 3) crypto sync loss is hard to detect, and slow to recover from, 4) Key Management is hard, slow & sometimes cryptos get mis-keyed, 5) Diagnostic tools are limited, and 6) When Out of Sync or improperly keyed, encryptors spew garbled plaintext into end equipment and/or application software, fouling their state machines and interrupting service. The use of a proper synchronization strategy minimizes this unreliability.

5.2.1.2.3 Encryption Module Logic Design

The basic logic design of the cryptographic module is shown in Figure 20. The crypto module receives plaintext cells in UTOPIA II (16 bit) format from the RAE shell, along with the appropriate index to the cryptographic state table as determined by the shell’s pre-processing of the cell header. The encryption module then converts incoming cells from the 16 bit-wide format to a 64 bit-wide format in order to reduce the clock rate required for processing, and to be compatible with the block width of the nonlinear cryptographic block cipher to be used. For this project, the 64 bit width also matches the block width of the Data Encryption Standard (DES) algorithm used.

The Control Processor initializes the contents of the context memory to contain the appropriate cryptographic key and initial state variables at the time the virtual circuit is set up.

The encryption module retrieves the cryptographic key and state variables associated with the virtual circuit with which the incoming cell is associated. The cell, which is divided into 64 bit blocks, is then encrypted by the encryption engine. Control signals coincident with the cell header information cause the encryption engine to pass the header information unencrypted while encrypting the cell payload information. The resulting cell with plaintext header and encrypted payload is converted from the 64 bit-wide format back to the 16-bit wide UTOPIA II format. It is then output to the shell for recombination with other cell streams from other adjacent encryption modules. In addition, the context memory is updated with the appropriate new cryptographic state to be retrieved for processing of the next cell to arrive in this virtual circuit cell stream.

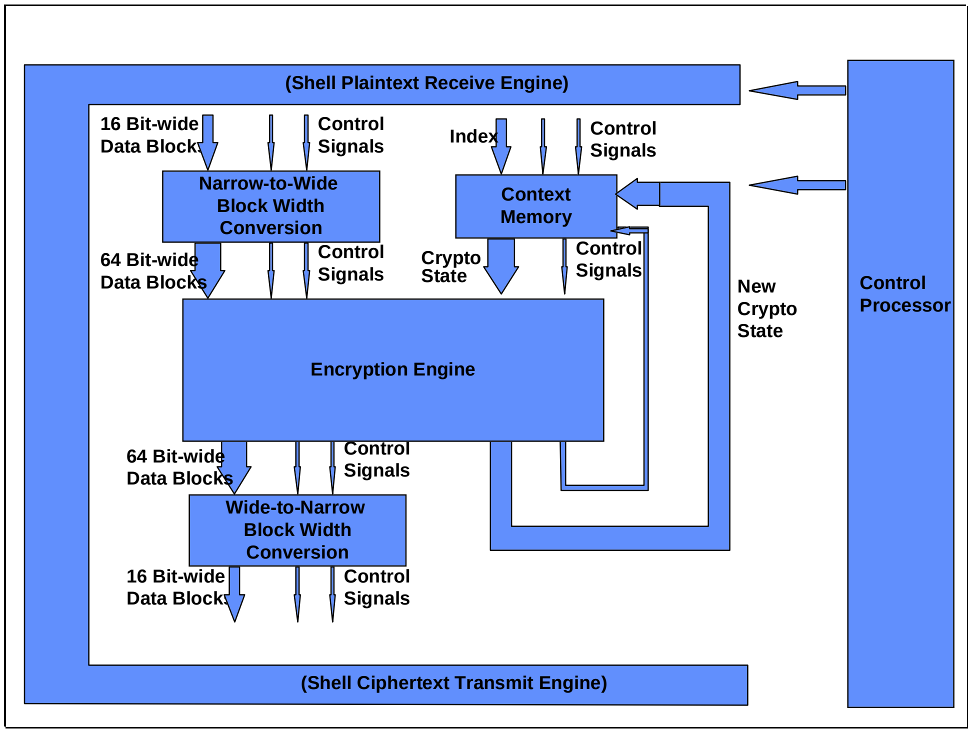


Figure 20: Encryption Module Logic Design

5.2.1.2.4 Decryption Module Logic Design

The decryption module logic design is symmetrical to the encryption module design, as shown in Figure 21. The primary difference is that the decryption module accepts ciphertext cells from the RAE shell, along with the index of the appropriate decryption state to be retrieved from the context memory. In addition, the inverse of the nonlinear block cipher that was used to encrypt the cell is used for the decryption operation.

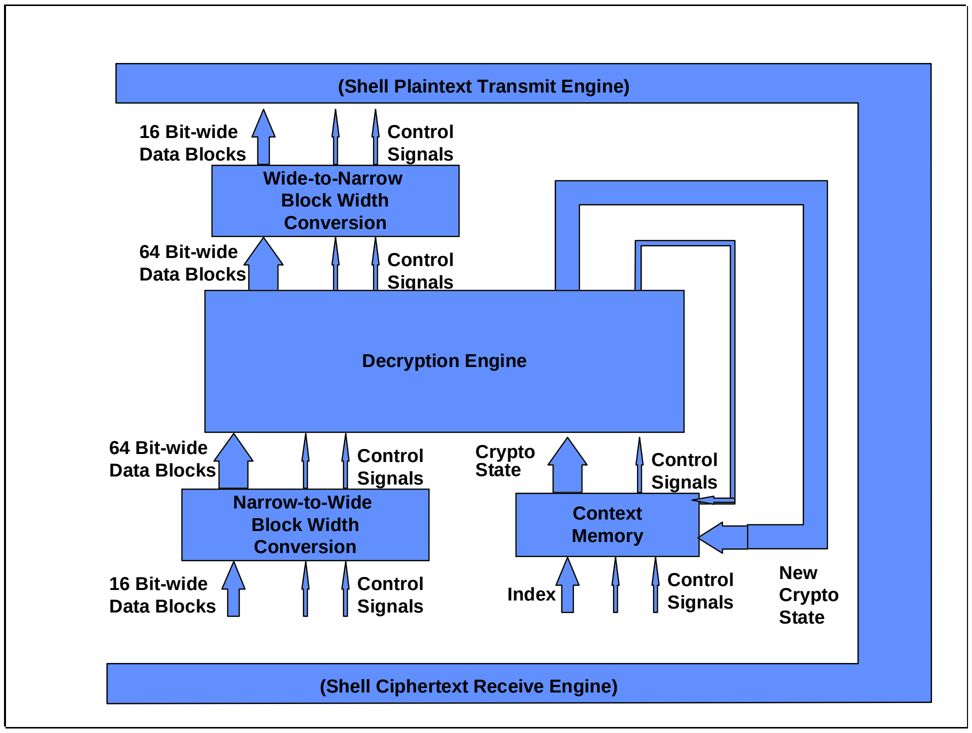


Figure 21: Decryption Module Logic Design

5.2.1.2.5 Encryption/Decryption Engine Logic Design

Although a single Encryption/Decryption Engine could be designed to configure itself to perform multiple algorithms and modes of operation, this research effort chose to implement a single algorithm and mode combination per crypto module. The RAE shell delivers cells to be encrypted or decrypted by a specific algorithm to the crypto module configured to perform that algorithm.

Therefore, Encryption/Decryption Engines are to be implemented with various different algorithms (nonlinear block ciphers) and with different “feedback modes of operation”. This section will describe the design of Encryption/Decryption Engines to implement counter mode and Cipher Block Chaining (CBC) mode of the same nonlinear block cipher algorithm, DES.

Figure 22 shows the logic design for a DES CBC Encryption/Decryption Engine. Again, CBC mode provides self-synchronization and also low implementation cost at low speeds. Since the computation of each succeeding block of ciphertext depends on the previous ciphertext block, the DES pipeline must “run dry” (can never be filled). This greatly increases the latency for each encrypted cell. This additional latency (with respect to other algorithms/modes) will cause increased Cell Transit Delay (CTD) or Cell Delay Variation (CDV), depending on the method used by the shell to combine the encrypted output cell streams from the various encryption pipelines.

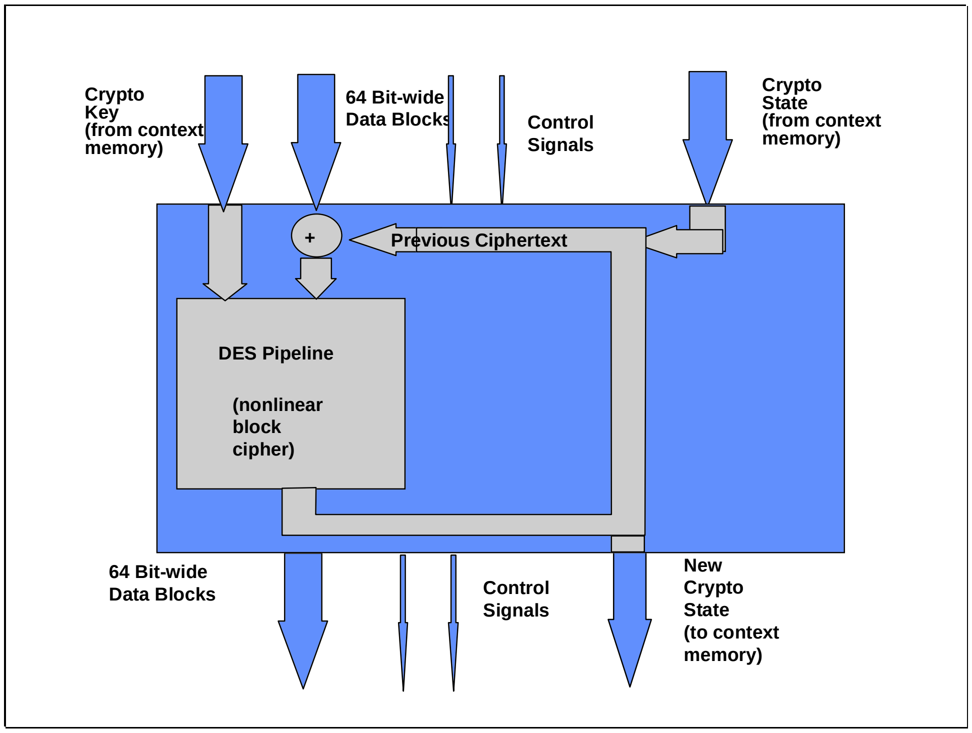


Figure 22: DES/CBC Encryption/Decryption Engine

Figure 23 shows the design of a DES Counter Mode Encryption Engine. This mode can be scaled efficiently for high speed parallel operation, and can be pipelined effectively to reduce cell latency. However, this mode requires some means of synchronizing the linear sequence generator at the decryptor with the one at the encryptor. The cost of implementing this synchronization method at low speeds is higher than for self-synchronizing modes such as CBC. Figure 19 showed this tradeoff in more detail.

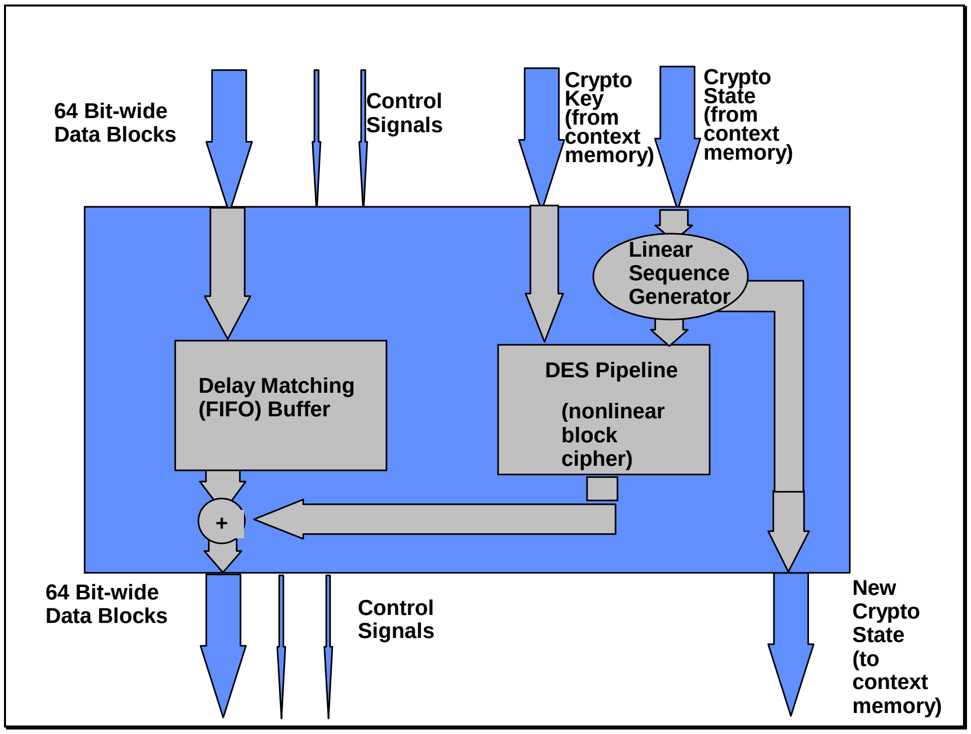

64 Bit-wide Data Blocks
Control Signals
Crypto Key (from context memory)
Crypto State (from context memory)
Linear Sequence Generator
Delay Matching (FIFO) Buffer
DES Pipeline
(nonlinear block cipher)
+
64 Bit-wide Data Blocks
Control Signals
New Crypto State (to context memory)

Figure 23: DES Counter Mode Encryption/Decryption Engine

Figure 24 shows a DES implementation which is pipelined for high speed operation and also augmented with a “bypass” control bit (to pass the plaintext cell headers).

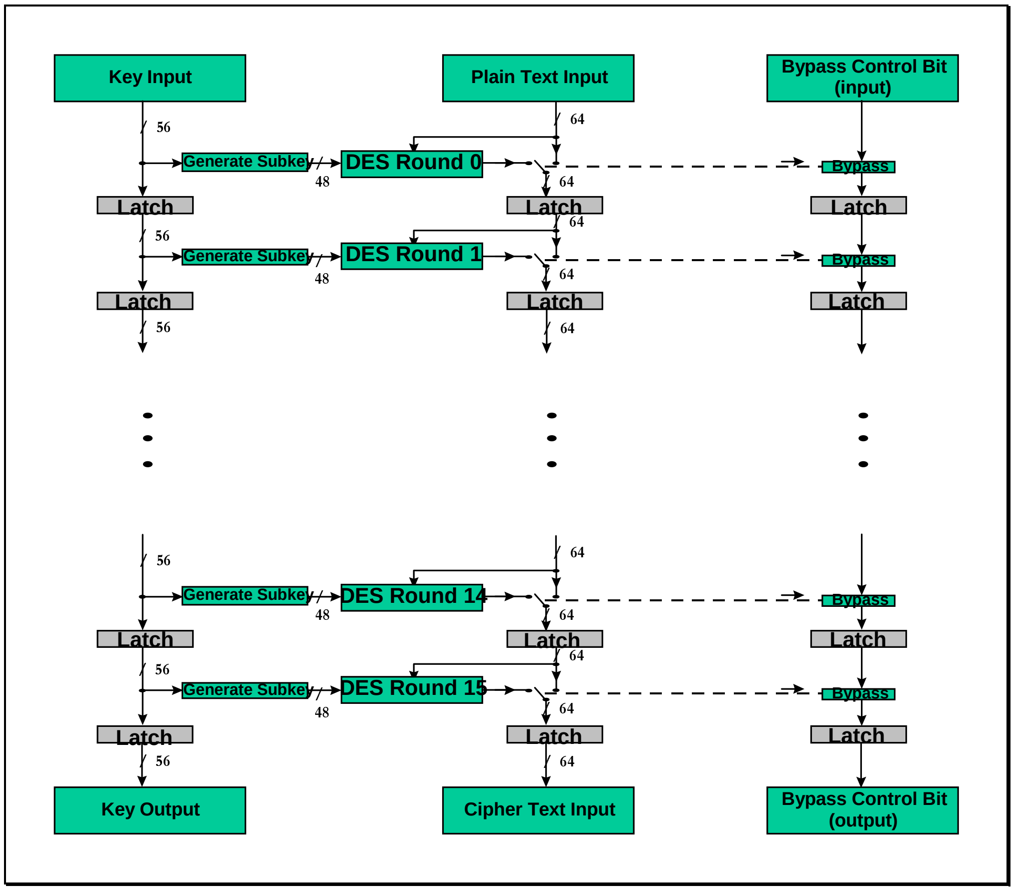


Figure 24: Pipelined DES Encryption Algorithm Implementation

5.2.2 RAE Software

The RAE software consists of four major components: the signaling module, the security agent, the hardware controller, and the smart card interfaces. The signaling module performs ATM connection setup signaling according to the ITU’s Q.2931 standard [7]. The security agent performs the ATM security protocols (e.g., authentication and key exchange) according to the draft ATM Forum Security Version 1.0 specification [1]. The purpose of the key management module is to provide secure storage of user and device keys. It also implements cryptographic functions for key exchange and authentication at connection setup. Finally, the hardware controller is responsible for configuring the encryption hardware according to the outcome of the authentication and key exchange protocols, and for monitoring the status of the various hardware components.

In order to implement the two-way security message exchange protocol described in Section 5.1, modifications of the host and RAE signaling software were required. The following figure shows the software architecture for the end user system:

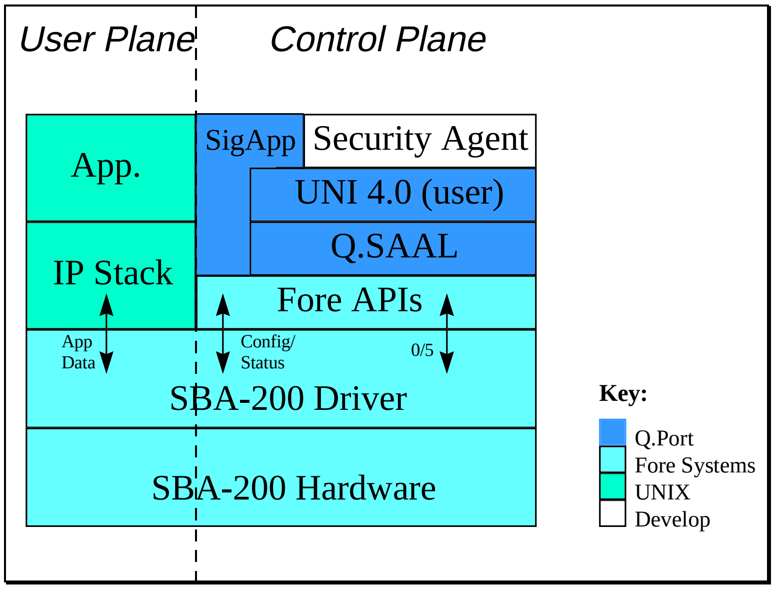


Figure 25: Host ATM Security Software Architecture

In the end system, most of the required software is already provided by Q.port, the Fore Systems drivers and APIs, and by the UNIX operating system itself. The only additional software required was the “security agent”, which implemented the two-way message exchange protocol summarized earlier. In addition, modifications to the UNI 4.0 (user-side) module were also required to pass the Security Services Information Elements to the Security Agent.

The following figure shows the software architecture for the RAEs:

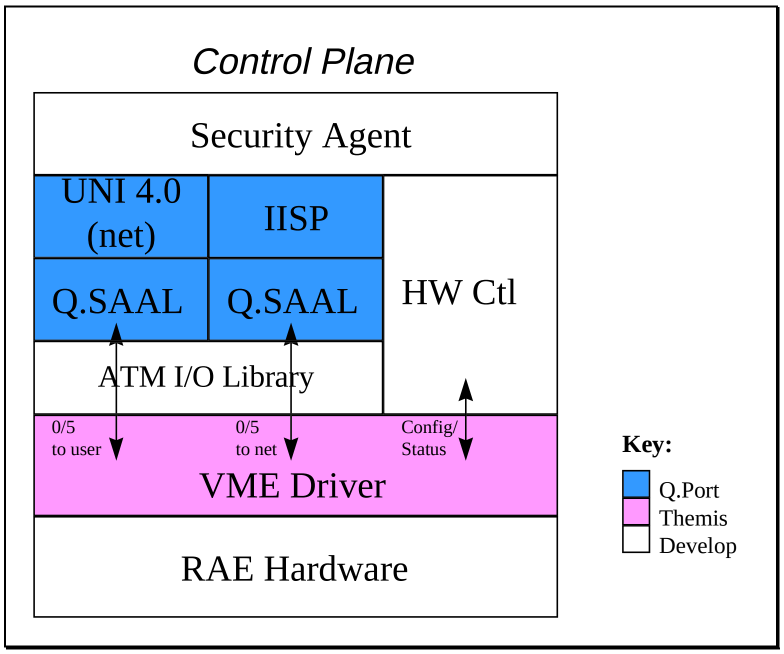


Figure 26: RAE ATM Security Software Architecture

As seen here, most of the RAE software was developed under this project. The only modules which were provided by other vendors were the Q.port software and the VME interface drivers. The software modules developed under this project include the Security Agent, the hardware control functions, and the ATM I/O functions. The RAE security agent is mostly identical to the host security agent, except for some minor configuration changes for security agent addressing and options for the key exchange service. The purpose of the hardware control module is to provide a mechanism which allows the security agent to configure the hardware as the security message exchange protocol progresses. In addition, since the security message exchange protocol is conducted through signaling, the ATM I/O library was developed to allow the signaling software to access the RAE transmit/receive registers for signaling traffic on VPI=0, VCI=5.

These four components (signaling, security agent, smart card interfaces, and hardware control) are implemented as two processes, as shown in the following figure:

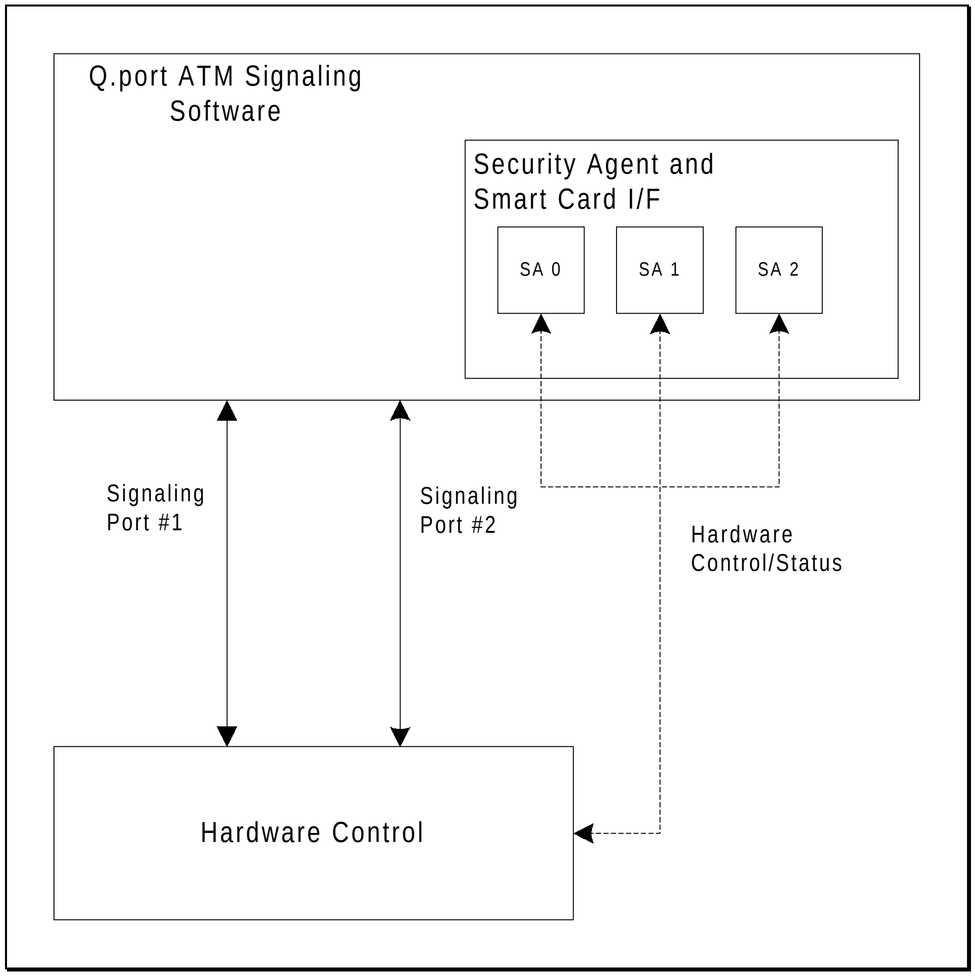


Figure 27: RAE Software Inter-Process Communication

Within the signaling process, messages are passed between the Q.port signaling software and the security agent in the form of calls to the security agent methods. Likewise, messages between the security associations (contained within the security agent object) and the card reader are passed in the form of function calls. However, because the hardware controller is implemented as a separate UNIX process, an inter-process communication mechanism is required to exchange hardware configuration messages with the security association objects.

The use of UNIX IPC mechanisms for communications between the “key management module” (in this case, the smart card) and the encryption hardware raises a potential vulnerability. This vulnerability exists because the traffic encryption keys are passed “in the clear” through an untrusted medium, i.e., the UNIX operating system. This vulnerability can be resolved in two ways – either harden the UNIX operating system so that this operation becomes “trusted”, or treat the operating system as an untrusted channel, and encrypt communications between the (trusted) key management module and the (trusted) encryption hardware. Hardening the operating system, while possible, is difficult due to its complexity. Furthermore, it may not be possible to harden the operating system to the point where it can be trusted to carry very sensitive keying material. Although the second approach requires custom hardware to encrypt messages between the KMM and the cryptomodule, it is preferred for three reasons. First, these modules are already presumed trusted. Second, this solution is simple to analyze. Finally, this solution does not require modifications to the complex UNIX operating system.

5.2.2.1 Signaling Software

The signaling software for the RAE is largely based on the Q.port signaling package from Bellcore. This package is an implementation of the ITU Standard Q.2931 [7] and the ATM Forum’s UNI 4.0 signaling protocols. These protocols specify the signaling required on the user to network interface to establish on-demand virtual circuits, or Switched Virtual Circuits (SVCs). In addition, this software also provides an implementation of the ATM Forum’s Interim Interswitch Signaling Protocol (IISP), which performs switch-to-switch signaling and call routing functions.

In order to transport security information within UNI 4.0/Q.2931 signaling, the Q.port software was modified to support the new “Security Services Information Element (SSIE)”, and the procedures associated with its use (see [1] and [3]). This information element contains the information fields required by the security agent when it carries out the security message exchange protocol. These fields include timestamps, digital signatures, and nonces required for authentication, encrypted keys for key exchange, and negotiation parameters. The Q.port signaling software required modification so that the SSIE would either be forwarded with the entire message to the security agent or, if there is no security agent, transported without modification. The format of the SSIE and its associated signaling procedures are defined in [3].

With respect to ATM signaling, the RAE functions as an ATM switch. That is, it implements the “network-side” ATM signaling on the ports which are connected to end systems, and it implements IISP on the ports which are connected to other switches or encryptors. However, the ATM switching “fabric” in the RAE is different from typical ATM switches in that it modifies cell payloads instead of cell headers. As a result, unlike typical switches, the RAE performs no VPI/VCI translation. For this reason, a new Fabric Control module for Q.port was developed which guaranteed that the same VPI/VCI values are used on the input and output ports.

5.2.2.2 Security Agent

The security agent is the software module which implements the security message exchange protocol specified in [1]. For each security association (i.e., each active or pending security service between this RAE and some remote RAE), there exists a security association object which implements the security message exchange protocol. In addition to implementing the protocol, these security agent objects also contain the negotiated parameters for the association, and they are responsible for communicating security-related configuration information to the hardware control module.

This module interacts with the Q.port signaling software via a set of “hooks” which were developed in Q.port for processing the SSIE. When the security agent receives a “process SSIE” request, it must first determine whether a security association (SA) already exists, using the “Security Agent Identifier” fields in the SSIE. If the security association exists, then the SSIE is forwarded to the appropriate SA object. Otherwise, a new SA object is created, and the SSIE is forwarded to that object. When the SA object receives the SSIE, it processes it according to the current state of the security message exchange protocol. If a processing error is encountered (e.g., authentication failure), the SA object returns an error code to the security agent, which returns an error code to the Q.port software (which then denies the connection request). Otherwise, at the conclusion of the security message exchange protocol, a success code is propagated back to Q.port, which then accepts the call. Also upon success of the protocol, the SA object instructs the hardware controller (via the IPC mechanism described earlier) to enable the hardware for the new virtual circuit, and to load the ATM cell encryption keys.

5.2.2.3 Hardware Control

The hardware control module is responsible for processing configuration requests from the security agent, and/or from an X Windows user interface. Functions performed by this module include enabling virtual circuits, loading keys, initiation of resynchronization, and transmitting/receiving signaling messages.

The X Windows interface is implemented in the Tool Command Language, with the X Windows Toolkit (TCL/Tk). The TCL script is responsible for managing lists of active virtual circuits, and for processing inputs from the user interface and/or the security association objects. It is also responsible for periodically polling the RAE hardware to check for changes in its status, and to check for incoming cells on the signaling virtual circuit (VPI=0, VCI=5). In order to perform hardware-specific functions, a set of “extended” TCL commands, written in C, provide access to the hardware registers over the VMEbus.

If signaling cells are encountered on either of the RAE’s two ATM interfaces, the hardware control code reassembles them into an AAL5 Service Data Unit (SDU). Once the SDU is fully reassembled, it is forwarded to the Q.port signaling software via the IPC mechanism described earlier. Conversely, if a signaling message is received from the Q.port software, then the hardware controller segments it into ATM cells, and transmits the cells out of the appropriate RAE ATM interface.

5.2.2.4 Smart Card Software

The smart card software developed for the RAE allows the security association objects to access the smart card during security message exchange protocol processing (at connection setup). The smart card has two functions in this prototype – secure key storage (for both user and device keys), and cryptographic processing.

A Smart Card is not unlike the ubiquitous credit card. Its physical dimensions and properties are much the same. But, in addition to containing data to be read by the reader to which it's introduced, the Smart Card also contains an embedded microprocessor. This allows the smart card to carry out its basic principal activity, specifically, the reception of a command from the terminal and the sending of an appropriate response from the card.

Communication between terminal and the smart card is always initiated by the terminal, which also supplies power to the card. Only one data line is available, and the communication transmission is therefore half-duplex: i.e., only one device may transmit at a time. One protocol with which we're familiar, specifies that the right to send always passes to the receiver of the last block.

The single chip microcomputer contained in the smart card is the Siemens SLE44C40, which has the following features:

- 256 Byte RAM (Working Area)
- 8 kByte ROM (Operating System)
- 4 kByte EEPROM (Applications and Data)

This configuration delivers 10 year data retention properties. It also allows a minimum of 100,000 write/erase cycles per EEPROM page with 1 page equal to 4 bytes. Data formats of 1, 4 and 8 bits are supported.

Throughput is limited by a maximum clock frequency of 7.5 MHz.

Access to the smart card's cryptographic functions occur through its StarCos operating system. This operating system controls access to "files" (which may contain keys) via user-entered Personal Identification Numbers (PINs). However, access to these files is restricted to "write only". (Although some files, specifically "purse files" have "execute" permissions, allowing such files to be securely incremented and decremented.) Therefore, these files are used for key storage, and can be configured such that these keys can only be accessed when a user successfully provides a PIN. This allows the user to be bound with the card, thereby providing user-level access control to the RAE.

The cryptographic functions provided by the smart card include digital signatures and encryption. Digital signatures, which are used for authentication, are performed using the Digital Encryption Standard Message Authentication Code (DES-MAC). Encryption, which is used for exchanging ATM cell encryption keys, is performed using DES in Cipher Block Chaining (CBC) mode. Although these algorithms are specified in the ATM Forum's security specification [1], they are not implemented precisely in the specified manner. For example, the smart card implements a different method of padding. Therefore, this implementation is not compliant with the ATM Forum's specification. However, other smart cards which implement digital signatures and encryption in a compliant fashion do exist, particularly those which implement "public key" algorithms.

6. Conclusions

Robustness-agile ATM encryption allows users and organizations to invoke flexible security services upon request. ATM users need that flexibility because they often have simultaneous virtual circuits to several different users in several different organizations. Those different organizations may have different security policies. Hence, ATM users may need to encrypt their connections with a variety of encryption algorithms and key lengths. In that case, a single ATM encryptor, that implements multiple algorithms, has several benefits. First, it reduces the management costs. Second, it eliminates the requirement for non-standard security policy-based routing mechanisms.

This report described the design, development, and issues associated with Robustness Agile Encryptors (RAEs). The RAE used software components to implement security services that were invoked at connection setup (i.e., ATM signaling, security protocol processing, and hardware configuration). Its hardware components then implemented security services during the duration of the data virtual circuit (i.e., cell routing and multiplexing functions, and cryptographic functions). Finally, other implementation issues included context lookup, effects on ATM Quality of Service (QoS), and access control.

7. References

[1] The ATM Forum Technical Committee, *ATM Security Specification*, *Version 1.0*, Straw Ballot, STR-SECURITY-01.00, The ATM Forum, 2570 West El Camino Real, Suite 304, Mountain View, CA, December, 1997.

[2] The ATM Forum Technical Committee, *Traffic Management Specification, Version 4.0*, af-tm-0056.000, The ATM Forum, 2570 West El Camino Real, Suite 304, Mountain View, CA, April, 1996.

[3] The ATM Forum Technical Committee, *UNI 4.0 Security Addendum*, ATM Forum BTD-SIG-SEC-01.00, The ATM Forum, 2570 West El Camino Real, Suite 304, Mountain View, CA, February, 1997.

[4] The ATM Forum Technical Committee, *UTOPIA Specification, Level 1, Version 2.0*, ATM Forum af-phy-0017.000, The ATM Forum, 2570 West El Camino Real, Suite 304, Mountain View, CA, March, 1994.

[5] The ATM Forum Technical Committee, *UTOPIA Specification, Level 2, Version 1.0*, ATM Forum af-phy-0039.000, The ATM Forum, 2570 West El Camino Real, Suite 304, Mountain View, CA, June, 1995.

[6] Ellison, Frantz, and Thomas, *Simple Public Key Certificate*, Internet Draft, Internet Engineering Task Force, March, 1997.

[7] The International Telecommunications Union, *B-ISDN DSS2 User-Network Interface Layer 3 Specification for Basic Call/Connection Control*, Recommendation Q.2931, February, 1995.

[8] L. G. Pierson and E. L. Witzke, *Multiply-Agile Encryption in High Speed Communication Networks*, SAND97-1069C, Sandia National Laboratories, Albuquerque, NM, April, 1997.

[9] L. G. Pierson, T. D. Tarman, and E. L. Witzke, *Scalable End-to-End Encryption Technology for Supra-Gigabit/second Networking*, SAND94-1622, Sandia National Laboratories, Albuquerque, NM, May, 1997.

[10] P. E. Sholander, *et. al.*, *The Effect of Algorithm-Agile Encryption on ATM Quality of Service*, Sandia National Laboratories Report SAND97-0489C, Proceedings IEEE Globecom ’97, pp. 470-474, Phoenix, AZ, November, 1997.

[11] T. D. Tarman, *et. al.*, *Final Report for the Protocol Extensions for ATM Security Laboratory Directed Research and Development Project*, SAND96-0657, Sandia National Laboratories, Albuquerque, NM, March, 1996.

DISTRIBUTION:

1		Jeff Ingle National Security Agency Attn: R222, R&E 9800 Savage Rd. Ft. Meade, MD 20755-6000
1	MS 0188	C. E. Meyers, 4523
1	MS 0188	LDRD Office, 4523
1	MS 0431	S. G. Varnado, 6200
5	MS 0449	R. J. Granfield, 6236
5	MS 0449	R. L. Hutchinson, 6236
5	MS 0449	P. E. Sholander, 6236
1	MS 0449	R. S. Tamashiro, 6237
10	MS 0449	T. D. Tarman, 6236
1	MS 0622	J. F. Jones, 4600
5	MS 0806	L. G. Pierson, 4616
1	MS 0806	M. R. Sjulin
1	MS 0806	M. O. Vahle, 4616
5	MS 0806	E. L. Witzke
1	MS 0815	L. L. Fine, 5931
5	MS 0874	P. J. Robertson, 1342
1	MS 9003	D. L. Crawford, 8900
1	MS 9011	H. Y. Chen, 8910
1	MS 9011	P. W. Dean, 8910
1	MS 9018	Central Technical Files, 8940-2
5	MS 0899	Technical Library, 4916
1	MS 0161	Patent and Licensing Office, 11500
2	MS 0619	Review and Approval Desk, 12690 For DOE/OSTI